Computational Design of an Integrated Learning and Assessment Platform for Higher Education

Dissertation

submitted to the Faculty of Business, Economics and Informatics of the University of Zurich

to obtain the degree of Doktor der Wirtschaftswissenschaften, Dr. oec. (corresponds to Doctor of Philosophy, PhD)

presented by

Maik Meusel

from Germany

approved in October 2019 at the request of

Prof. Dr. Karl Schmedders

Prof. Dr. Thomas Lontzek

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, 23.10.2019

Chairman of the Doctoral Board: Prof. Dr. Steven Ongena

Acknowledgements

I am very grateful to my supervisor Karl Schmedders, for his teaching, guidance, and support on this project. With his boundless patience and mettle in creating testing opportunities he accelerated the project in all its aspects.

For continuous fruitful technical discussions and guidance I would like to thank Nick Portokallidis, Jakub Podkalicki, Carlos Chida, and Rolf Mertig. I am equally grateful for conceptual discussions on the subject and mental support from Kenneth Judd, Jessica Sudo, Anand Sundaram, Kathrin Kuhn, Jan Henne de Dijn, Thomas Lontzek, Christina Richard, and Maximilian Franke.

For this project, receiving substantial technical support from Wolfram Research Inc. was vital. In particular I would like to thank Stephen Wolfram, Conrad Wolfram, Jesús Hernández, and Nate Pillow for their encouragement and collaboration.

For their valuable feedback and patience in testing over the past years I would like to thank Roman Mäder, Jasmin Maag, José Parra-Moyano, and hundreds of students.

Contents

List of Figures

List of Tables

List of Abbreviations

3D	Three Dimensional
AESG	Applied Educational Services Germany GmbH
AI	Artificial Intelligence
API	Application Programming Interface
app	Application
ASCII	American Standard Code for Information Interchange
Avg	Average
B.Sc.	Bachelor of Science
BYOD	Bring Your Own Device
CA	California
CD	Current Date
CDF	Computational Document Format
CentOS	Community Enterprise Operating System
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSV	Comma Separated Value
CV	Curriculum Vitae
DB	Database
DevOps	Development and Operations
DoS	Denial of Service
Dr.	Doctor
e.g.	Exempli Gratia
eBook	Electronic Book
ed.	Edition
ED	End Date
Email	Electronic Mail

etc	Et Cetera
Feb.	February
FinTech	Financial Technology
HTML	Hypertext Markup Language
https	Hypertext Transfer Protocol Secure
i.e.	Id Est
IBM	International Business Machines Corporation
iframe	Inline Frame
ILAP	Integrated Learning and Assessment Platform
img	Image
Inc.	Incorporated
IT	Information Technology
JSON	JavaScript Object Notation
LaTex	Lamport TeX
lhs	Left-Hand Side
LMS	Learning Management System
M.A.	Master of Arts
M&A	Mergers & Acquisitions
MA	Massachusetts
MAGIC	Massive Automated Grading in the Cloud
Max	Maximum
Min	Minimum
MOOC	Massive Open Online Course
n/a	Not Applicable
NB	Notebook (Wolfram System notebook format)
NLP	Neuro-Linguistic Programming
No	Number
NoSQL	Not Only Structured Query Language
oec.	Oeconomiae
OLAT	Online Learning and Training
PD	Publish Date
PDF	Portable Document Format
Perf	Performance
PhD	Philosophiae Doctor

pm	Post Meridiem
Prof.	Professor
RBAC	Role-Based Access Control
rhs	Right-Hand Side
RWTH	Rheinisch-Westfälische Technische Hochschule
SD	Start Date
SVG	Scalable Vector Graphics
Tech. rep.	Technical Report
TLS	Transport Layer Security
USA	United States of America
UI	User Interface
UK	United Kingdom
URL	Uniform Resource Locator
UX	User Experience
UZH	University of Zurich
vs.	Versus
WEPC	Wolfram Enterprise Private Cloud
www	World Wide Web
XML	Extensible Markup Language

Chapter 1

Introduction

The history of higher education goes back centuries and has experienced innovations caused by social, economic, and technological changes throughout that time. Once a luxury good for a small and privileged fraction of society, it has become accessible to the majority of people all over the world in recent decades. In fact, and not for postindustrial societies alone, it is nowadays the main driver for innovation and wealth.

Surprisingly, the higher education system itself lags behind in adopting the innovations it has, itself, identified. While in many industries the digital transformation is well advanced—even in the financial sector, older and more conservative, with the rise of digital transactions, crypto currencies, and "financial technology" (FinTech)—most exams conducted today remain paper-and-pencil affairs.

This thesis focuses on the recent emergence—with its opportunities and challenges—of digital transformations and provides a concrete, practical, yet comprehensive solution approach for higher education. The key understanding here is that technological innovations can enable and lead necessary transformations in other fields, such as politics and pedagogy.

1.1 Problem and solution

In pursuit of this goal, I will argue that computer-based education has the potential to innovate three aspects of education:

- Liberate hierarchical structures in higher education by introducing a third-party authority.
- Use interactivity to reshape the entire learning journey by closing the gap between abstraction and concreteness.
- Improve learning success directly and indirectly by means of automated assessment and evaluation.

From a knowledge to a certification monopoly—to collaborative education

Historically educators have been the monopolists of knowledge. In times when recording and conserving knowledge was both difficult and expensive, universities were the only places to access it. Over the course of centuries this monopoly position has been eroded by several innovations, starting with the invention of printing and continuing through the development of library systems to the advent of the Internet. Today, knowledge is almost a public good, accessible to the majority of people.

With the decline of this earlier monopoly a new—even stronger—one has emerged. While today knowledge is available to all, no matter where one comes from, holding a degree from a reputable university is still the gold standard for every CV. The initial focus of universities on research has shifted toward a much broader purpose—training and education. In many professions a university degree is indispensable, either explicitly by law or implicitly in society's terms. One of the main contributions that has led to this monopoly of what we call the *modern* university can be seen in the Humboldtian model of higher education in the early nineteenth century (von Humboldt, 1997). Since then, the holistic combination of research and education has been adopted by all leading universities around the world—and is still in place. We now observe that this monopoly is being challenged in several ways: On the one hand, new institutions are providing online and distance learning, including certification, at much lower prices and with modern, technology-driven learning approaches. On the other hand, the academic job market has become more competitive resulting in higher pressure for knowledge creation and cost efficiency in teaching. While the demand for education is still growing (Calderon, 2018; Docebo, 2016), students are more critical with regard to the value proposition of a traditional university education.

In the light of this discussion one may ask whether the certification monopoly will be eroded in the same way as was the knowledge monopoly, or put simply: What is next? While it may still be too early to answer this question with certainty, the direction is already visible. The teacher–student relationship has always been characterized by an imbalance of knowledge and power with issues arising similar to those of principal–agent problems; that is to say, in our case, the incentives for learning and teaching are not aligned.

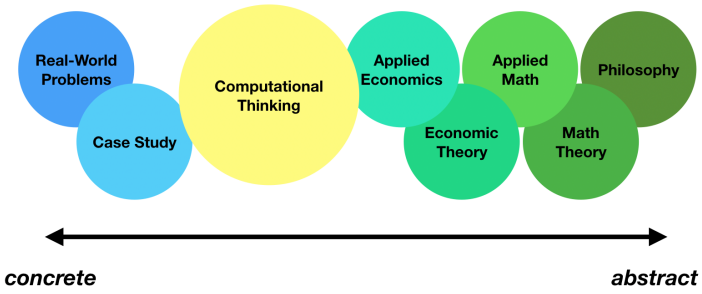
In this analogy, educators take the role of the *principal* (note the equivocalness) and act as effort minimizers while bearing the bureaucratic burden as administrators of learning. Students, in their role as *agents*, act as grade maximizers by assimilating the demands and the weaknesses imposed by educational institutions and their educators. While this system has been in place for centuries, it is not necessarily the optimal framework for efficient learning. This can be resolved by taking away some of the responsibilities of educators and giving them to a third—independent and trustworthy—party. By taking

over the enforcement of rules and—to some extent—sovereignity with regard to evaluations, educators gain time and freedom, and can redefine their role as that of engaging in the active mentoring and support of students' learning paths. Thus, by dismantling the hierarchy the traditional role model erodes, benefitting a new—*collaborative*—approach to higher education (Scager et al., 2016).

From interactivity to computational thinking

Research is often about finding abstractions, detecting patterns in complex subjects, and generalizations. While this is undoubtedly the foundation of our modern knowledge ecosystem and has led to innovation and progress in all academic fields, it is not necessarily the best approach for teaching. Building intuition, exploring, and simply trying things for oneself are often the most effective way of learning new concepts (Ouadoud et al., 2018). In contrast to research, which is still predominantly *static* in terms of how knowledge is condensed and published, all these actions are *interactive*.

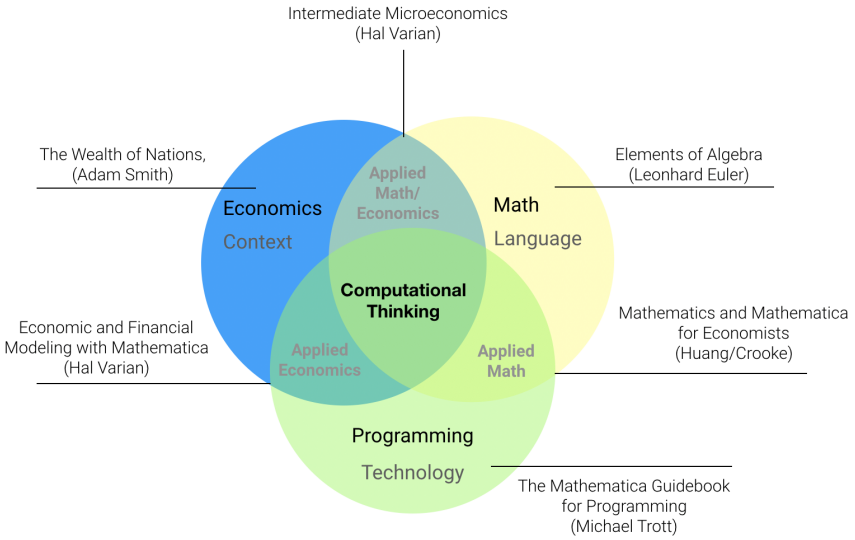
Figure 1.1.1: Levels of abstraction in teaching Business and Economics



Note that interactivity is not only an aesthetic feature employed to make materials more appealing, but—if used in a meaningful way—can help to overcome the hurdles of understanding new concepts by closing the gap between the concrete and the abstract. To illustrate this idea consider the academic field of Business and Economics. In Figure 1.1.1 we can see an attempt at ordering with regard to constituent elements' level of abstraction. On the one side *case studies* are used to teach business administration as they are closest to real-world problems. On the other side, economics is usually taught in a highly abstract and mathematical way. These teaching approaches are often perceived as incompatible antagonisms although they have the same underlying theories.

Computational thinking[1], an approach that allows one to apply computational (abstract) knowledge to computable (real-world) data by means of heavy automation with regard to the required computations (Wolfram, 2016b), has the potential to close this gap.

Figure 1.1.2: Computational thinking and interdisciplinarity



However, in order to do things interactively, and to enable computational thinking, the right tools are needed—tools that are not yet used in education. Integrated tools provide the potential to enable interdisciplinarity in teaching all kinds of subjects (Wolfram, 2016b). In Figure 1.1.2 we see overlapping academic fields and that there is usually an *applied* version of every pure discipline that makes use of another field. Current textbooks often lie at the intersection of two disciplines, but all of them are lacking in terms of combining all relevant dimensions. Computational thinking can fill this gap as it allows one to incorporate all the concepts into one teaching approach by means of interactivity.

From teacher-driven to learner-centered assessment

As summarized in a famous quote by Thomas Carlyle—"No pressure, no diamonds"—assessment is the key to learning. One can have libraries full of books, but to actually internalize one's understanding requires some means of interactive challenge—the assessment. What is necessary for students has, however, become the biggest burden for educators. Manual grading of assessments—beside

[1]The term "Computational Thinking" was discussed initially by Wing (2006), Bundy (2007), and Wing (2008). Applications such as those supported by SYLVA are discussed in Wolfram (2016b).

its tedious and monotonous nature—can be the most time-consuming task educators face. As a consequence, assessments are often designed with the goal of being easily gradable. In doing so, either questions are forced into cramped templates, or heuristics such as grading on a curve are applied (Krumboltz and Yeh, 1996). Both distort the original educational purpose of assessments—that is, to reinforce learning concepts. Because of the high costs (time) for educators, the number of assessments is reduced to the minimum required by educational institutions for certification purposes. This, however, makes things more risky for students. Failing an exam can—at some universities—mean that students have to wait a full year to retake it, and that they lose all the effort they have already invested. In fact, this leads to high workloads in the exam period, as opposed to continuous studying activities. All this contributes to a learning-to-the-test culture (Rojstaczer and Healy, 2012; Babcock, 2010), which is not sustainable (Huba and Freed, 2000).

We have been used to this system for decades, but what would happen if educators were released from the burden of grading? What if it was effortless to evaluate hundreds of answers, and to reevaluate them should errors occur? It would free up time for educators to rethink their assessment strategies, and likely to better support their students to succeed.

SYLVA—An Integrated Learning and Assessment Platform

The solution offered in this thesis to the aforementioned problems is an Integrated Learning and Assessment Platform (ILAP) called *SYLVA*. It allows teachers and students to use a single platform for all educational affairs by integrating the core features—that is, authoring, courseware, assessment, evaluation, and management—into a coherent system of apps, as shown in Table 1.1.1.

1.2 Scope and limitations

Main contributions

The contributions of this thesis can be summarized in four main points:

1. The thesis presents a functional realization of a higher education product that covers all educational purposes from authoring, to distribution, assessment, and evaluation, brought together in a single platform, the ILAP.

2. It derives a general, comprehensive, and extendable condition- and rule-based approach for automated grading.

3. It provides a coherent and systematic approach for standardizing the naming, definition of units, and formulation of processes in computer-based assessment and evaluation.

Table 1.1.1: SYLVA apps overview

Logo	Name	Purpose
	SYLVA	Integrated Learning and Assessment Platform (ILAP)
	Courseware	Viewing course materials
	Assessment	Taking assessments
	Evaluation	Viewing assessment evaluation reports
	Administration	Managing courses and students
	CREO	Authoring of courseware and assessments

4. It enables computer-based education approaches such as teaching computational thinking.

Limitations and remarks

The ideas and solution approaches proposed in this thesis are to be seen as *one* way of approaching the aforementioned issues and challenges by means of an ILAP—without any assertions that SYLVA is the *only* or *best* way of its implementation. The design of a platform is a complex process that involves numerous components, including the computational design, the technical implementation, the UI and UX design, and many more. While SYLVA presents a novel approach of combining a technical computing system (developed by Wolfram) with modern web technologies, neither of these were developed within the scope of the project. Similarly, neither the idea of an educational platform[2] nor the automation of grading is new[3]—it is the combination of these that, supplemented by a comprehensive assessment and evaluation approach, makes SYLVA unique. It is, therefore, important to note that the focus of the thesis is on the integration of computational and educational concepts and technologies—as opposed to their technical implementation.

[2]LMSs have been around for more than a decade; their use and enhancement has been widely researched and discussed, with some controversy. For examples refer to Dalsgaard (2006), Vrasidas (2004), Aydin and Tirkes (2010), Coates et al. (2005), Adzharuddin and Ling (2013), Yueh and Shihkuan (2008), and Chung et al. (2013).

[3]Automated grading for higher education is researched and discussed in various disciplines; for examples refer to Fox et al. (2015), Van Dalen et al. (2015), Shermis (2014), Wilcox (2016), Zhang (2013), and Geigle et al. (2016).

Starting with a single Mathematica notebook to grade one exam at the University of Zurich in 2013, SYLVA has grown to a fully self-serviced educational platform. This was an iterative and agile development process in which feedback from educators and students, several (unrecorded) conversations with educators at international conferences, and each experience gained by using it for teaching were taken into account at each stage to improve SYLVA. This evolution of the product is not portrayed in the thesis, nor are future plans and possible extensions discussed. Instead, this thesis attempts to give an extensive overview of the current state of the platform and its ability to tackle challenges faced by students and teachers alike. In doing so—due to the significant complexity of the platform—not all features can be explained in every detail[4] of their use; instead we will focus on innovative features, setting aside typical platform functionalities.

SYLVA, in its current and previous versions, has been used for the teaching of more than 1,000 students between 2013 and 2019 in the following courses at the University of Zurich:

- Applied Empirical Methods for Business Administration (2013–2019)
- Applied Business Modelling and Analytics (2015–2019)
- Computational Economics & Finance (2015)
- Mathematik 1 (2016)
- Programming Bootcamp (2018–2019)
- Blockchain for Managers (2018)

While many of its features have already passed the testing and prototyping stage, it is still to early to be able to evaluate the effectiveness of SYLVA with regard to its impact on improving learning success. So far, only a few small-scale surveys have been conducted, which—while positive in sentiment—do not satisfy scientific standards or have enough statistical power to enable us to draw reliable conclusions yet. Therefore, these surveys are not discussed in this thesis. Note too that for data privacy reasons no results or underlying data from the courses taught can be made available.

The project this thesis concerns has received substantial funding in the form of technology and financial grants from Wolfram Research International, IBM, and Google, and was implemented and financed by AESG. While the ideas and concepts presented in this thesis are not protected by patents of any of the aforementioned companies, the technical implementation of SYLVA, including all its source codes, procedures, and graphical designs, as well as all related copyrights and marketing rights are the sole property of AESG, and are therefore not part of this thesis.

[4] An extensive collection of learning resources including examples for most of the features discussed in this thesis is publicly available (AESG, 2019).

1.3 Notation and structure

Notation

All codes and functions that are explicitly mentioned and discussed in this thesis are Wolfram Language (Mathematica). Functions are emphasized as Function. Evaluatable codes are written as in the following example: `Table[i^2, {i, 10}]`. Neither the codes nor the functions are derived or explained in detail in this thesis. References to functions can be found at https://reference.wolfram.com. For readers of the digital version of this thesis, all functions are linked to the particular pages of the online documentation. Readers who have no prior knowledge of the Wolfram Language are referred to Wolfram (2016a), and Hastings et al. (2015).

All figures and graphs are my own creations or based on my own calculations. Figures that show components of SYLVA, or any results generated by it, are approved, for publication in this thesis, by AESG. Some of the figures in this thesis are related to courses I taught at the University of Zurich and, therefore—while not explicitly showing them—are based on sensitive data that cannot be made publicly available. Without loss of generality, all such examples can be replicated in SYLVA with different data. Graphical representations of SYLVA UIs and related features are due to change in the future. The figures shown as well as the features explained in this thesis are based on the following versions of the particular products:

- Wolfram Mathematica 11.3.0.0 (released March 8, 2018)
- Wolfram Enterprise Private Cloud 1.47.2 (released October 15, 2018)
- CREO 6.20.1 (released May 17, 2019)
- SYLVA Platform 2.2.3 (released June 20, 2019)
- SYLVA API 2.0.4 (released May 21, 2019)

Structure

This thesis attempts to cover all aspects of an ILAP by deriving requirements for relevant features, before explaining how to use them, and is structured as follows: Chapter 2 starts with a discussion of requirements for computer-based course materials (Section 2.1). In the subsequent sections, emphasis is put on the authoring and deployment of courseware (Section 2.2) using CREO, and on collaboration between educators in creating and distributing (Section 2.3) such materials in SYLVA. Section 2.4 introduces scheduling of courses and lectures.

In Chapter 3, initial specific requirements for computer-based assessment and evaluation are derived as a basis for further discussions in subsequent sections. Assessment properties, options, and types (Section 3.2) build the basis for creating assessments in SYLVA. In Section 3.3 we explain—by discussing

several examples—how various types of questions qualify for computer-based assessment, and discuss implications for formalizing and evaluating these questions. Based on these conceptual considerations, in Section 3.4 we construct five concrete questions and explain their setup in CREO. In Section 3.5 these example questions are used for an in-depth simulation of the entire evaluation process. Section 3.6 finishes with reporting of evaluation results.

Following the elaboration of the core concepts, in Chapter 4 selected topics of administration specific to ILAPs are examined. The final chapter provides insights on the technical challenges—focusing on computational aspects—related to using SYLVA for higher education, and how they are addressed.

Chapter 2

Courseware

2.1 Requirements for computer-based courseware

In this section we will discuss the specific requirements for creating and distributing interactive courseware.

2.1.1 Availability and usability

While computer-based assessments are still not used in most higher education institutions, platforms for sharing and distributing courseware have been around for many years and are widespread. One of the reasons for this broad dissemination can be seen in the convenience of online materials, which can be easily accessed via standard web browsers. SYLVA intends to provide a convenient solution for both students and educators. Most of the current LMSs, including Moodle, Blackboard, and OLAT (in Switzerland) are based on the idea of file sharing—that is, educators upload their materials as files, usually PDFs, to the platform and are then able to share them with students. However, the materials are usually still created using specific tools such as Microsoft PowerPoint, Microsoft Word, or LaTex editors (Dahlstrom et al., 2014; Arvan, 2009). While this gives educators a lot of flexibility with regard to the tools they can use, the approach has some disadvantages too.

First, the files containing the materials have to be organized and stored separately as they cannot be edited within the platforms. This usually requires further file sharing tools like Dropbox, in particular when the materials are created or managed in collaboration between several educators. While many users are familiar with those tools, and actions such as sharing a file are not time-consuming, it still takes effort to set up and maintain a system for collaboration.

Second, using static files for courseware always entails compromises since there is no one-size-fits-all version of courseware (Kroner, 2014; Dahlstrom et al., 2014). For in-class lectures, the educators usually prefer some form of slides, which are optimized for projectors with regard to their layout and content. In-

stead of longer explanatory texts, educators often prefer a more reduced version of the contents to accompany their in-class presentation and discussion. In order to share more detailed explanations they need to create and distribute a separate *reader* or *script* version of the materials or refer to books. While this is not the most convenient solution, even if they created those additional materials the underlying problem would not be solved. A slide show usually does not have a layout that is optimized for reading (self-study or post-processing). As a consequence, students often print all the slides (on possibly hundreds of pages) just to get a better overview of the materials. More generally the problem is that course materials are not responsive; that is, depending on the device the student is using (tablet or laptop), the materials won't be convenient for study purposes. To overcome this problem, web pages (HTML) can be used as the main format for courseware, while supplemental file downloads for offline use can be offered.

Third, to create a smooth learning journey for students, it is important to integrate all course components, in particular courseware and assessments (Kroner, 2014). Using HTML-based courseware allows one to link or embed components in the most flexible way. In addition, the updating process for courseware becomes much easier: instead of creating new files, storing different versions, and upload them again, web pages can be updated with a single click with no interruption or broken dependencies.

2.1.2 Interactivity

When we look at the history of science (and higher education), for centuries books and papers have been the medium for storing and sharing knowledge. One of the reasons for this can be seen in the invention of printing, which made it possible to copy (and distribute) documents easily and cost-efficiently. Of course, at the time, modern inventions such as recording audio or video or creating software were not available, leaving no alternative.

Books and papers are *static* documents. That means they consist of texts and graphics—such as images or tables—that cannot change or react to their readers once they have been printed. While the PDF is nowadays the most common format for creating and sharing documents *digitally*, the initial problem has not changed: the documents are still static. However, static documents are not necessarily the best learning tools (Somers, 2018). First, it is hard (in some cases impossible) to visualize dynamic processes or movements via a static document. Second, the process of learning often involves tasks such as exploring, trying, and replicating—which, again, are difficult or impossible to realize with a static document (Ouadoud et al., 2018). Third, to foster understanding (the goal of learning), interaction—that is to say, communication between the learning resource and the learner, to check and verify comprehension—is necessary (Chen et al., 2010), but static documents cannot provide this. Fourth, static documents can only be consumed through reading, which for some peo-

ple might be cognitively harder than listening or watching, and undoubtedly is more monotonous than addressing a combination of different senses.

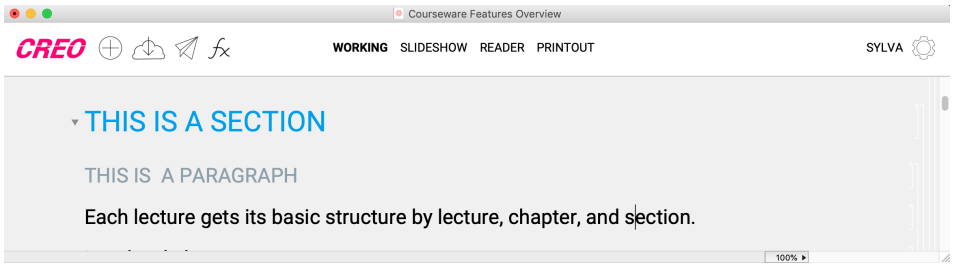
To address all the shortcomings mentioned above, a superior alternative nowadays exists: software, in particular a web application, is capable of combining static content, audio, video, and all kinds of interactions. So why are PDFs still the most commonly used medium? With recent technological advancements the costs of copying and distributing software are almost the same as those for static documents. The reason can be seen in the creation process. While it is easier than ever to create video and audio content with devices such as a smart phone, creating, distributing, or hosting software is still inconvenient or too expensive for most educators. One of the trends in education in recent years is the MOOC, which offers many of the relevant features mentioned above (Docebo, 2016). Note however, that these online courses target thousands or even millions of users, rather than a single course for a few (hundred) students at a particular university. Some of these courses are commercially distributed on platforms such Courseare or Udacity, or depend on donations, for instance at Khan Academy. The costs of creating such software solutions can only be covered when the content is distributed on a large scale.

SYLVA attempts to enable individual educators to create software-based courseware for *any* targeted class size. By providing an authoring tool that is integrated into the platform, all deployment and distribution tasks can be automated to the extent that educators do not need software development skills or resources. In this way, SYLVA is offering an alternative to MOOCs—the individual, private, blended-learning course—to address the shortcomings of online-only courses (Bettinger et al., 2017; Baker et al., 2016). In doing so, one of the core principles is to enable interactive course materials as the main learning resource for students. To make this possible SYLVA is built on third-party software, functions, computational knowledge, and content developed or hosted by Wolfram Research. This allows educators to use demonstrations, tools, data, and code for interaction directly within the course materials.

2.2 Authoring and deployment

In this section, first authoring is explained in detail; we then go on to discuss the deployment and collaboration. To create courseware with SYLVA there is an authoring tool, called CREO, which is a an add-on to Wolfram Mathematica. Figure 2.2.1 shows the basic authoring notebook in Mathematica. The core idea is to build on an extensive existing programming language to allow users to take advantage of all the functions and computational knowledge developed and hosted by Wolfram Research.

Figure 2.2.1: Courseware editing UI



2.2.1 Basic styles

CREO is used with the Mathematica notebook interface, which is a document containing cells as its lowest level elements. With hundreds of options, a cell can be customized with regard to the layout, style, and behavior in terms of the notebook and evaluations. It is important to note that, in contrast to CSS, cell options determine the functionality of a particular cell, including for example if and how the content is evaluated, and not just the formatting—that is, how it is displayed.

We will start by looking at the basic styles for educational purposes. Note that while the styling and many specific features have been customized in CREO, it is still entirely build in Mathematica and adopts the main concepts of cells.

Lecture

The *lecture* style is the highest order structural element of a courseware notebook. All other cell styles will group under the lecture style. A courseware notebook must have at least one lecture cell but can have several lectures. For deployment, however, each lecture in CREO will be associated with exactly one lecture in SYLVA. A CREO notebook with several lectures can be deployed to several lectures in SYLVA, but not merged.

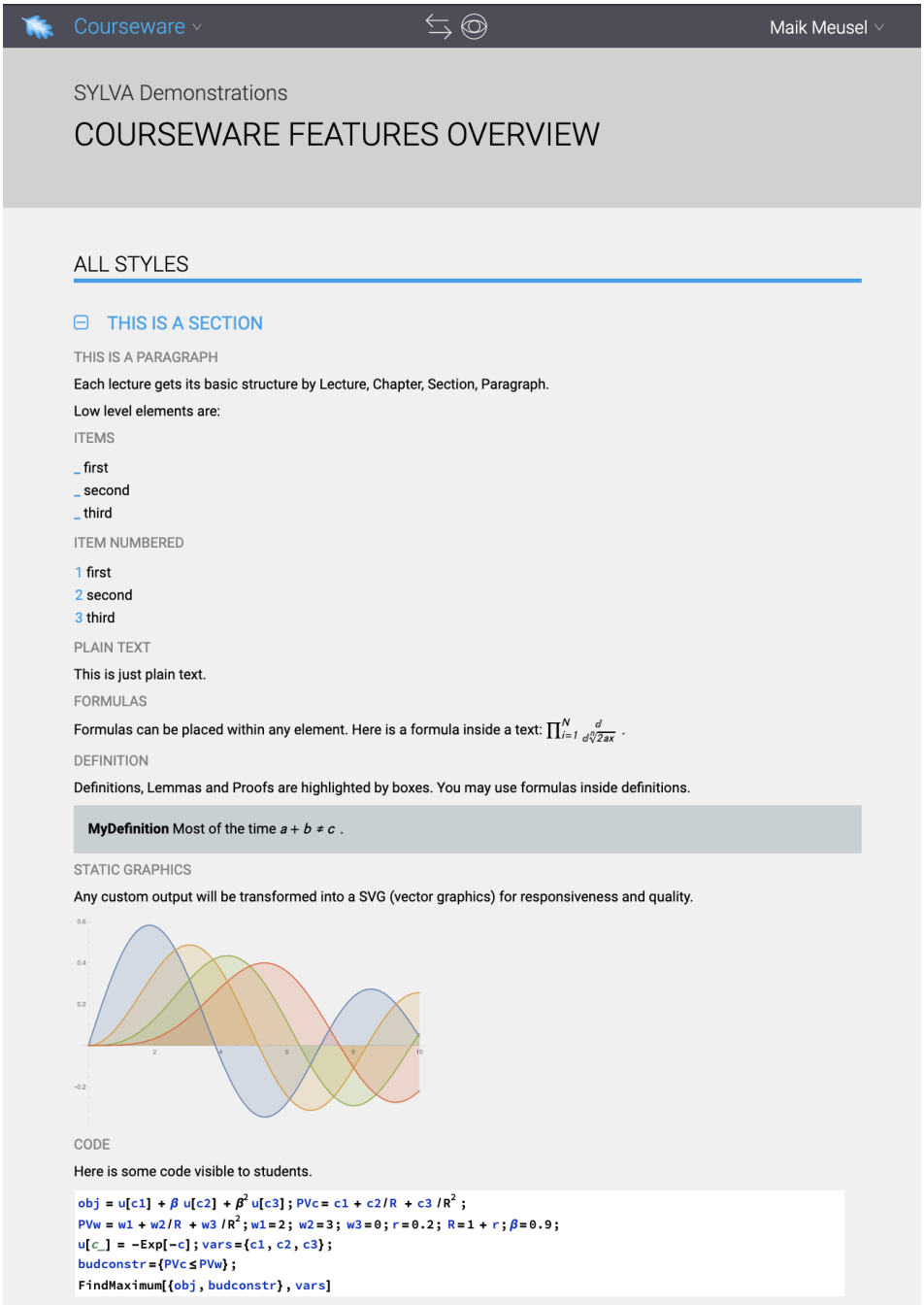
Chapter

The *chapter* style is the second order structural element of courseware notebooks. Each lecture must contain at least one chapter. This style defines the parts of a lecture that will, by default, be deployed to separate slides in the slideshow version.

Section

The *section* style is the lowest order structural element of a notebook. Following the hierarchy of styles, sections belong to a chapter. However, they are optional

Figure 2.2.2: Courseware styles overview



elements, so a chapter can have sections. When deployed to online courseware, all contents within sections are grouped and collapsed with openers.

Paragraph

The *paragraph* style does not have a specific structural function. Paragraphs are therefore atomic style elements that can be flexibly used and combined within sections or chapters. The main purpose of paragraphs is to visually divide longer texts by serving as subheadings.

Text

Another atomic style is the *text*. As the name suggests, this style can be used for regular text. Longer texts can be organized either in separate cells or within one cell as line breaks are supported.

Explanation

Despite a different appearance, the *explanation* style can be used just like the text style. However, explanations are only shown in the reader version of the course material—that is, they are hidden from the slide show. With this style educators can create different versions within one file in the same working mode. This is particularly useful as it allows students to review the lecture content on their own (for instance if they are unable to attend the lecture), or for a blended learning approach. As mentioned in the requirements in Section 2.1.1, in a slide show (for in-class presentation) the material is usually used to supplement the lecturer's explanations; for example a graph showing some data while the lecturer is explaining the relationship between the data in some context. Not only would it take the attention of the students away from the lecturer's explanations if all the explanations were already shown on the slide, it would also restrict the layout if one tried to fit all the content on a single slide.

With explanations educators can overcome these issues without having to create two separate versions of the same material. In the current implementation of SYLVA this style is restricted to textual content, but a more flexible implementation (as an option for all styles rather than a style itself) will be implemented when the deployment methods advance further.

Definition

The *definition* style allows authors to highlight important concepts—such as formulas or propositions—visually. Beside the specific styling, definitions are otherwise similar to text cells.

Item

The *item* style is designed for unordered lists. Each item is marked by a dash (bullet). Cells with this style are grouped together and by default typing Enter adds a new item.

Item numbered

Just like items, the *item numbered* style can be used for listing items. All cells are grouped together and numbered starting at one. The numbers replace the dashes of simple items. The numbering is consecutive within a particular cell group only—that is to say, a separate cell (group) will restart with number one.

Input

Input cells are the default cell style for adding new cells in Mathematica. For authoring in CREO, additional styling has been added to support the consistency of the layout and the appearance of the UI. The behavior of input cells has otherwise not been modified. This allows educators to flexibly copy and paste input cells from existing Mathematica notebooks, and vice versa. Input cells can be deployed to allow students to see, modify, and evaluate code directly within the course materials.

Output

Output cells, by default, are created by evaluating input cells. When this happens, a cell group is created to link the output to the input it was created from. Then, modifying and reevaluating the input overwrites and updates the output cell. Analogously to input cells, this default behavior has not been changed in CREO. Again, only adjustments of the styling have been applied to support the UI.

Hidden code

The *hidden code* style is designed to omit input cells from being deployed and displayed. This is useful when the educator does not want to reveal the code, or when it isn't relevant to the content. Imposed on input cells, the style adds a frame around the code, but preserves all other properties of these cells.

Static graphic

The *static graphic* is an imposed style that determines the deployment option and changes the style, typically for output cells. It can be applied to any atomic style, but removes the specific styling in some cases.

Cells that are marked as static output will deploy to a static visual representation of the content. The most common usage is to transform an output cell,

for instance a plot, into a static graphic. The plot will then be represented as an SVG in the online version of the courseware. Note that some Mathematica outputs, such as 3D plots or anatomical sculptures, are (visually) dynamic by default. The static graphic style can also be used to force (degrade) a dynamic expression into a static representation.

Dynamic graphic

The *dynamic graphic* style can be imposed on any interactive expression, such as those generated by Manipulate, DynamicModule, or 3D objects. Again, this style determines the deployment option. In contrast to static graphics, a dynamic graphic will deploy to an embedded cloud object, which enables the interactivity needed to make it dynamic.

Video

As video content is becoming more common in learning materials, SYLVA offers the opportunity to embed third-party hosted videos in the courseware. Currently links from Vimeo and YouTube can be pasted into CREO using the *video* style. Videos are not embedded into Mathematica notebooks or PDFs, only into the web version. Instead, clickable links are displayed whenever a video cannot be embedded directly. Note that currently SYLVA offers neither video creation nor hosting solutions.

Comment

The *comment* style is designed for authoring purposes, in particular when educators collaborate. It can be used for notes or remarks. Therefore comments are not deployed—that is, they are never visible to students in the course materials.

Inline formula

For quantitative fields in particular the course materials will typically include mathematical notation such as formulas. While Mathematica allows one to create mathematical symbols with additional palettes, these special characters or notations can usually not be displayed as plain strings in HTML. However, markup languages offer the flexibility to define tags for such cases. This is important when mathematical notation is combined with regular strings (text), for example. To enable this flexibility (Mathematica is not a markup language), mathematical notation can be injected into cells (styles) as an inline cell (with different properties). While such inline cells could be detected automatically when deploying the course materials, for authoring purposes this is not the best solution. First, inline cells could be created for purposes other than mathematical notation. Second, as most scientific documents are created using TeX, educators often want to copy and paste mathematical notation from there. To

support these different use cases in CREO, the *Inline formula* style is implemented as an additional button in the menu bar. Authors can either use the typesetting assistants and apply the style by clicking the button afterward, or before, or click the button and paste a TeX expression. In either case the inline formula content will be highlighted to be distinguishable from regular expressions or other inline cells.

2.2.2 Courseware authoring environments

CREO is neither a markup language editor nor a "what you see is what you get" tool, but rather a mix of both. This means, on the one hand, that the deployed version of the document looks (at least slightly) different than it did in the state in which the document was created. On the other hand, this also means that the deployed versions don't need to be compiled. CREO allows educators to switch between the different states or versions of the material. To give authors a real-time preview of all different versions of the material, CREO offers four different environments: working, slideshow, reader, and printout. To illustrate and explain the environments we will use a minimal example to compare the differences.

Working environment

The working environment, as shown in Figure 2.2.3, represents the editing stage. Compared to the other environments only the working stage has the following characteristic properties:

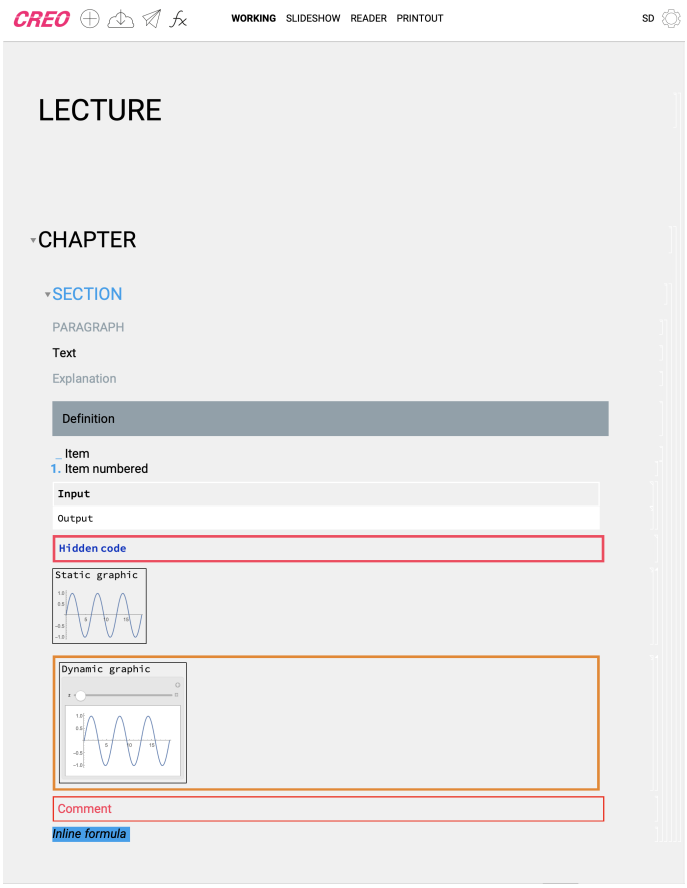
- All styles are visible and editable in this environment.
- All cell brackets are visible to make it easier to see the structure, in particular cell grouping.
- All generating and explicit code is visible and evaluatable.
- All styles are marked with specific colors to make it easier to distinguish them.

Slideshow environment

In Figure 2.2.4 we see the slideshow environment, which provides a preview of the Mathematica notebook version and the online slideshow version of the course material. Unless the author intentionally keeps input cells in the document, this version will not contain code, and therefore will look like typical slides created with other tools. Compared to the other environments, the slideshow has the following properties:

- A navigation bar containing the custom branding and logo is displayed at the top of the preview.

Figure 2.2.3: Authoring: the working environment



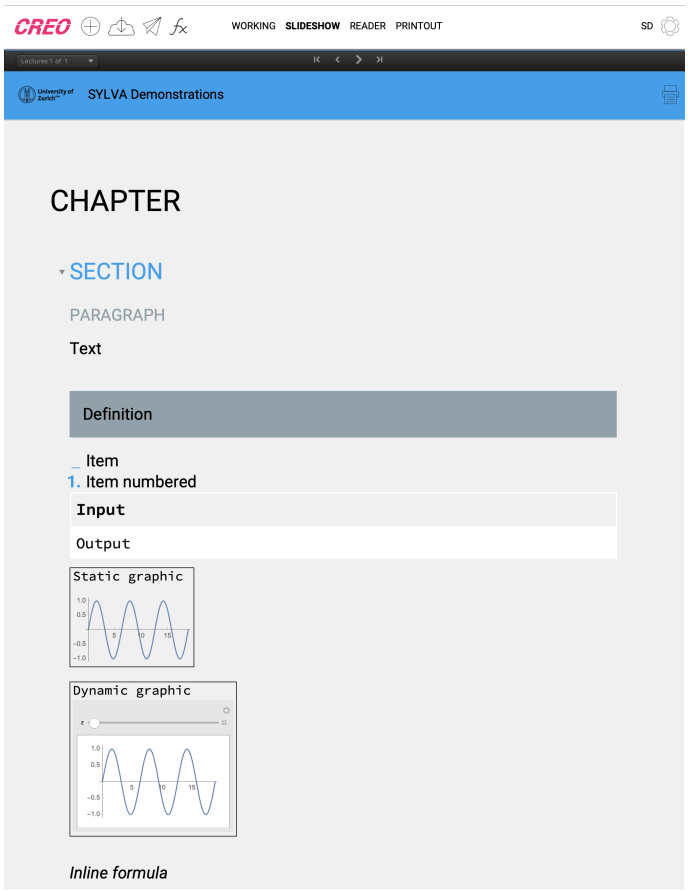
- The document is split into slides. By default the lecture (style) and each chapter are separated. The document still permits scrolling, but all sections are closed—that is, the content is only revealed once the section is opened.

- The following styles are not visible: hidden code, generating code of static and dynamic graphics, comment, and explanation.

- Only code that is explicitly marked as input (style) is evaluatable. Outputs are updated (overwritten) with every evaluation. Dynamic graphics are functional, so controls can be modified and updated outputs are shown.

- Cell brackets are preserved, but concealed. They have the same color as the notebook background and are only visible on hover.

Figure 2.2.4: Authoring: the slideshow environment

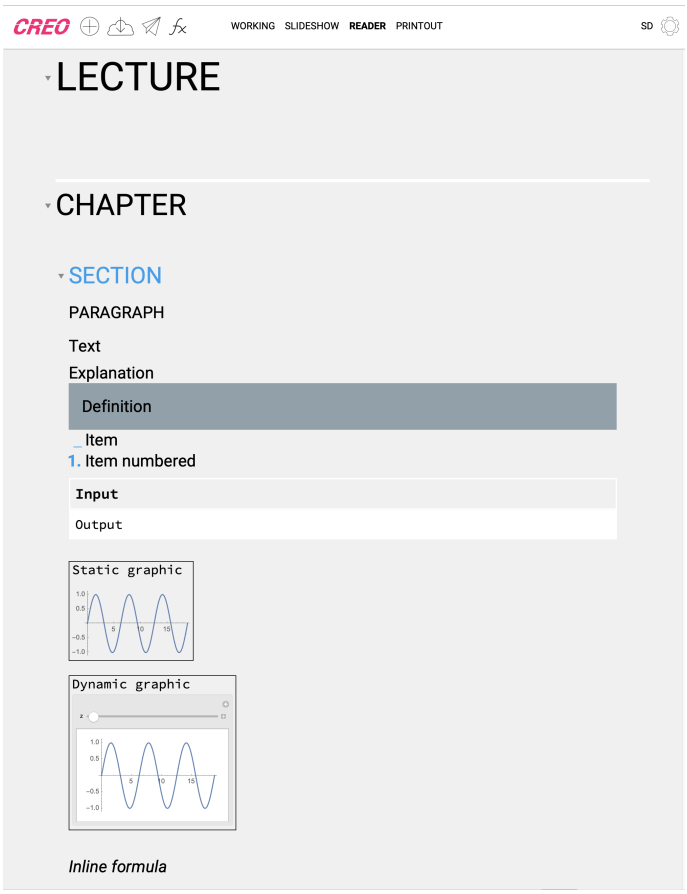


Reader environment

The reader environment, as shown in Figure 2.2.5, provides a preview of the more detailed reader (or script) version of the material. In contrast to slideshow this document is not split and is therefore comparable to an eBook. The specific characteristics of this environment are:

- The document is floating with all chapters and sections closed (default).
- The explanation (style) is visible and provides more details of the material. Otherwise all hidden styles from slideshow apply to the reader version too.
- Input cells are evaluatable but not editable. Dynamic graphics are functional.
- Cell brackets are preserved, but concealed.

Figure 2.2.5: Authoring: the reader environment

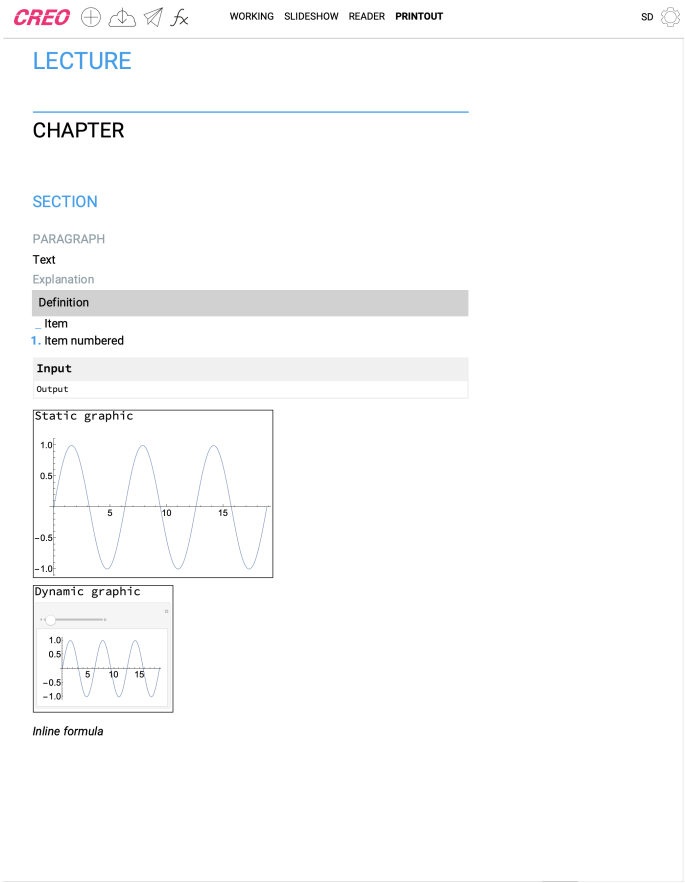


Printout environment

The printout environment gives a preview of the PDF version of the material, shown in Figure 2.2.6. As this version is static, it differs significantly from the other environments:

- The document is split by pages. All chapters and sections are expanded.
- Hidden code, any generating code, and comment styles are not visible.
- None of the styles is editable or evaluatable. For dynamic graphics a thumbnail is shown containing the UI elements, which are disabled however.
- Cell brackets are removed.
- The background (color) is removed. Other colors are reduced.
- Images for static and dynamic graphics are shrunk and the overall layout is more compact.

Figure 2.2.6: Authoring: the printout environment



2.2.3 Deployment options

Once the authoring process is complete, CREO offers three basic deployment options: web page (SYLVA), Mathematica notebook, and PDF. The last two are optional downloadable add-ons to the web deployment. Offering an online version, a computable (offline), and a static version of the materials gives educators and students the choice of using the materials in the most convenient way for them. In the following sections more detail on the options is given.

Web page deployment

Based on the document created with CREO a web page is created and associated with a particular lecture in SYLVA, accessible in the Administration app. Once the lecture has been accepted (see Section 2.3.2) and scheduled (see Section 2.4), the material will be available in the Courseware app, to students and educators (see Figure..). Optionally the Mathematica notebook and PDF can be downloaded here.

Just like in the CREO preview the web page provides the (more compact) slideshow and the (more detailed) reader versions for in-class and self-study purposes.

Table 2.2.1: Courseware styles overview

Style	**Hierarchy**	**Type**	**Deployment Options**			**Environments**			
			Online	Notebook	PDF	Working	Slideshow	Reader	Printout
Lecture	1	structural	•	•	•	•	•	•	•
Chapter	2	structural	•	•	•	•	•	•	•
Section	3	structural	•	•	•	•	•	•	•
Paragraph	4	atomic	•	•	•	•	•	•	•
Text	4	atomic	•	•	•	•	•	•	•
Explanation	4	atomic	•	•	•	•	–	•	•
Definition	4	atomic	•	•	•	•	•	•	•
Item	4	atomic	•	•	•	•	•	•	•
Item numbered	4	atomic	•	•	•	•	•	•	•
Input	4	atomic	•	•	•	•	•	•	•
Output	4	atomic	•	•	•	•	•	•	•
Hidden code	4	atomic	–	–	–	•	–	–	–
Static graphic	4	atomic	•	•	•	•	•	•	•
Dynamic graphic	4	atomic	•	•	–	•	•	•	–
Video	4	atomic	•	–	–	•	–	–	–
Comment	4	atomic	–	–	–	•	–	–	–
Inline formula	5*	atomic	•	•	•	•	•	•	•

Mathematica notebook deployment

In courses targeting coding, more computationally intense subjects, or in situations where there is no internet connection, Mathematica notebooks can be used for teaching and learning. In Figure 2.2.7 we see an example of such a notebook. These notebooks are computable documents similar to CDFs, both of which offer interactive computations. However, users will need either to have Mathematica installed (paid license required) or to use a free CDF player. In most cases using the notebooks is therefore a backup option.

The Mathematica notebooks contain the slideshow version of the course materials and allow the user to generate PDF (reader version) by using the print button in the navigation bar.

PDF deployment

The main advantage of PDFs is that they are applicable and printable on almost all devices and systems without the need to install additional software. The PDF, which can be downloaded from SYLVA or generated from the Mathematica notebook, contains the reader version of the courseware, optimized for printing. In Figure 2.2.8 we see an example fur such a document.

Figure 2.2.7: Courseware: Mathematica notebook

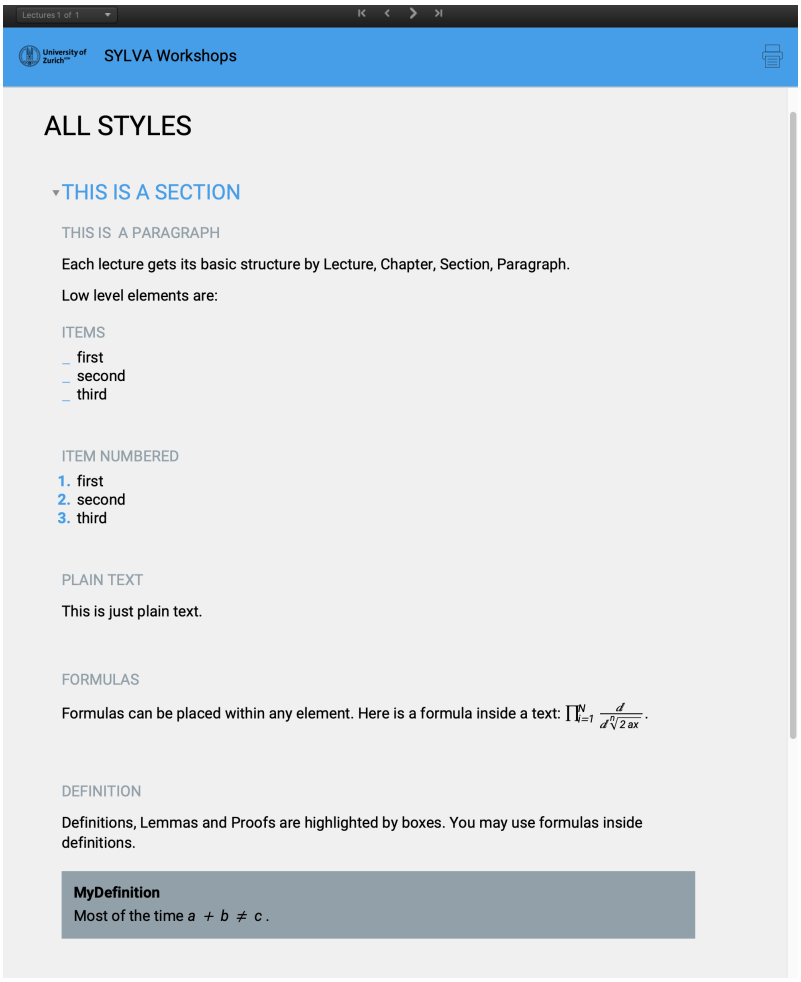


Figure 2.2.8: Courseware: PDF printout version

COURSEWARE FEATURES OVERVIEW

ALL STYLES

THIS IS A SECTION

THIS IS A PARAGRAPH

Each lecture gets its basic structure by Lecture, Chapter, Section, Paragraph.

Low level elements are:

ITEMS

- first
- second
- third

ITEM NUMBERED

1. first
2. second
3. third

PLAIN TEXT

This is just plain text.

FORMULAS

Formulas can be placed within any element. Here is a formula inside a text: $\prod_{i=1}^{N} \frac{d}{d\sqrt[n]{2ax}}$.

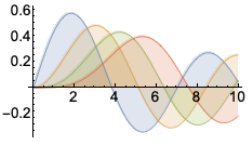
DEFINITION

Definitions, Lemmas and Proofs are highlighted by boxes. You may use formulas inside definitions.

MyDefinition
Most of the time $a + b \neq c$.

STATIC GRAPHICS

Any custom output will be transformed into a SVG (vector graphics) for responsiveness and quality.



Parsing notebooks

One of the advantages of using CREO to create course materials is that the content is natively structured and grouped in cells. Since all cells are associated with a particular style, this information can be used to distinguish pieces of content and place transformation rules, such as inclusion and exclusion in some versions, but also to parse the content into different languages.

In Mathematica all contents are represented by expressions contained in cells. While none of these expressions is natively using HTML, the Wolfram Language offers two solutions to transform notebook content into HTML: First, entire notebooks can be deployed as so-called cloud notebooks that are hosted on a Wolfram (Private) Cloud. Second, expressions can be converted into other formats and embedded in a generic HTML document structure. The first option is more convenient as all necessary transformations are handled by built-in Wolfram Language functions (and integration with the Wolfram Cloud). This approach, however, has a couple of downsides:

- Lower performance: The rendering of cloud notebooks is handled within the Wolfram Cloud and therefore consumes resources of the underlying infrastructure. This leads to lower performance (as more computational processes are performed server side[1]) and is harder to scale as user numbers grow.
- Less flexibility: With the convenience of staying within the Wolfram ecosystem comes limitations with regards to flexibility. Any custom element that is not generated in or supported by the Wolfram Language needs to be integrated into the cloud notebook framework, which is more complex than the specific structure of courseware pages in SYLVA.
- Limited styling options: Since the cloud notebook needs to be embedded into the platform, any change in the window size needs to be communicated to the iframe. Currently options for responsive styling are very limited and methods of communication with the iframe are lacking. Furthermore, the CSS for cloud notebooks is hard to customize, in particular for single elements and notebooks.
- Authoring technology dependencies: If the courseware content is created in a different language or tool, all content would need to be transformed to Wolfram Language expressions in order to be deployed. Due to the deep complexity of the language, this would mean more effort than adapting to courseware specific standards of SYLVA.

Given these limitations, the second option is less intrusive and more efficient. For the conversion each CREO style is translated to a HTML class and structure

[1] In the current version of the Wolfram Enterprise Private Cloud, client side rendering and computations are not supported. Furthermore, the rendering is less efficient than current standard applications.

first. As can be seen in Figure 2.2.2 this is straightforward since the hierarchy is already determined by the cell grouping in the CREO notebook. In a second step the actual expressions need to be transformed into elements that can be stored and interpreted outside the Wolfram Cloud ecosystem.

In most cases texts and letters are simply transformed to strings. For graphical elements such as output and static graphics a vector format, SVG, is chosen to preserve the quality and rescaling of the content. Note that a static graphic or output could be just texts or single characters. In those cases the content could be transformed into strings as well. However, this would require more detection and pre-processing efforts as there are various types of expressions that can be generated by the Wolfram Language.

In CREO, formulas are always used *inline* with other styles. Thus, whenever mathematical language is used, it will be inserted into another style by using the specific formula class tags. To display the actual notation, a JavaScript library, MathJax[2], is used, based on the transformed MathML input from CREO. Again, for the rendering only CSS and SVGs are used so that all mathematical notation scales like text and graphics.

Table 2.2.2: Courseware parsing overview

CREO	**HTML Classes**	**Format/Type**
Lecture	lecture/header/title	string
Chapter	chapter/header/title	string
Section	section/header/title	string
Paragraph	section-content/p	string
Text	section-content/text	string
Explanation	section-content/explanation	string
Definition	section-content/definition	string
Item	section-content/ul/li	string
Item numbered	section-content/ol/li	string
Input	section-content/coding	iframe [CloudObject]
Output	section-content/static-graphic-white	img [SVG]
Hidden code	*not deployed*	—
Static graphic	section-content/static-graphic	img [SVG]
Dynamic graphic	section-content/demonstration	iframe [CloudObject]
Video	/section-content/video	url [Player]
Comment	*not deployed*	—
Inline formula	*/formula	MathML

The (evaluatable) input and dynamic graphic styles typically contain Wolfram-Language-specific expressions that have no standardized independent format. Those cases are deployed as a so called Wolfram Language CloudObject, which can store and render any expression. For the styles mentioned above the cloud objects are usually reduced (simplified) cloud notebooks that are embedded as

[2]MathJax is a tool to display mathematics, based on JavaScript (MathJax, 2019).

iframes. This means that in SYLVA heavy cloud notebooks are only used when it is necessary to do so.

Finally, videos are embedded—based on the URL provided in CREO—using third-party players from Vimeo and YouTube.

Styling and branding

In the Wolfram Language the styling and functionality of notebook elements are determined by Stylesheets. While basic concepts like style inheritance, margins, and fonts are implemented in a similar way as CSS, Stylesheets cannot be transformed into CSS easily. The main reason for this is the variety of possible behaviors of Mathematica cells, which exceeds those of web browsers. Since the Wolfram Language was designed for a single application, Mathematica, the Stylesheets are not optimized for the deployment of single expressions or cells (but rather to optimize the performance of loading the entire application). As a consequence styles are organized hierarchically based on several levels of inheritance based on a few core styles. Thus, to transform Stylesheets into standalone CSSs, most of the complex style system of the Wolfram Language is required.

Another reason is the difference between certain units. While some units in Mathematica, like ItemSize have no standardized objective equivalence, relative units commonly used in CSS—such as *em*, *vh*, and *vw*—are not supported by the Wolfram Language out of the box.

As a consequence, styles from CREO are not deployed directly. Instead style schemes are implemented and communicated via the metadata in the deployment process. Corresponding CSS definitions have to be maintained to create a smooth transition between the two applications.

Currently, there are three style schemes implemented that users can switch between independently. In many cases users might prefer to adopt a predefined and consistent style over the option of customizing all details of the layout and appearance.

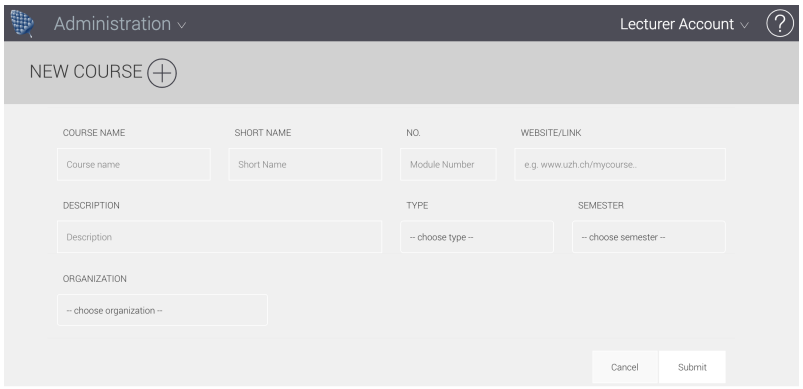
In SYLVA all projects inherit the branding, which is a style scheme and the logo from the organization. Thus, all projects associated with an organization will get a consistent and standardized styling without the authors setting or customizing styles at any point.

2.3 Collaboration and review

In many courses teams of educators work together on creating and distributing the materials. Particularly in larger courses teaching assistants usually take care of the exercises and of updating the courseware while the lecturer gives the class presentations and designs the exam questions. To facilitate such collaborations SYLVA offers flexible workflows to create, review, and deploy materials. Granular permissions help to attain different hierarchy and responsibility levels.

In the following sections we will cover all steps of the authoring and distribution process, starting with the project setup, content creation, and reviewing, then moving to scheduling and distribution.

Figure 2.3.1: Creating a new project



2.3.1 Creating in teams

Before the actual contents for lectures and assessments can be created a few initial steps are necessary to set up the project structure including the collaborators. In SYLVA every user who has been invited or has signed up as an educator can create projects from the Administration app, as shown in Figure 2.3.1. Creating a project makes the user owner of the project by default. In section 5.4 we will discuss roles and permissions in more detail.

Once the project has been created additional collaborators can be invited from the users tab in the Administration app (see Figure 2.3.2). Both roles (lecturer and teaching assistant) have permission to create lectures. After completing these initial steps, all collaborators have access to the project, and to each of its components in CREO. For every lecture or assessment a separate Mathematica notebook file is created via the window shown in Figure 2.3.3.

These files contain all UI elements needed during the authoring process, as well as links to and the metadata of the parent project. Since they are still regular Mathematica notebooks they can be saved locally for offline editing and sent to, or opened by, other users. However, the common authoring process involves saving the files to the cloud without the need for storing or handling them at all. Once a file is saved it will be automatically updated and available to all collaborators. At this point there is no version control implemented and simultaneous edits are always overwritten by the latest changes.

For communication between coauthors regarding parts of the content within the documents the *comment* style (mentioned in Section 2.2) can be used.

Figure 2.3.2: Inviting collaborators to a project

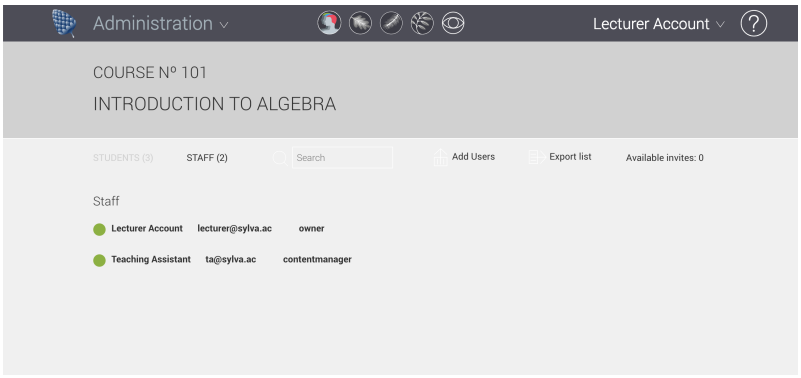
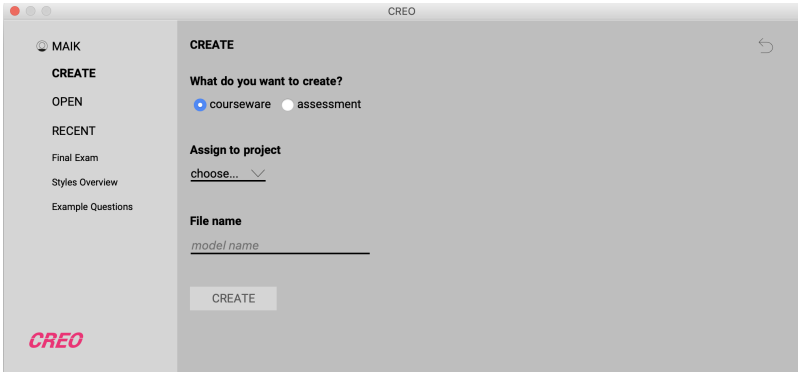


Figure 2.3.3: Creating course components in CREO



2.3.2 Review and proposals

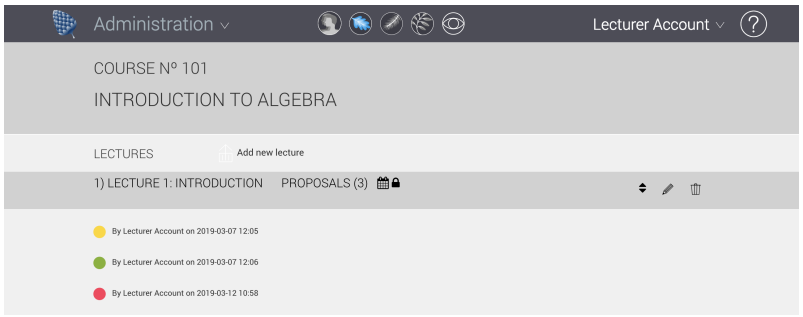
At any point in the creation process the materials can be deployed as a PDF file, Mathematica notebook, or the online version (2.2.3). While the first two options do not include the distribution of courseware, in the last case students would get direct and immediate access to the materials on the platform.

This, however, leads to the following risks: one of the co-authors could distribute an unconfirmed version; several coauthors could distribute several possibly conflicting versions of the same material. In the worst case this would lead to confusion among students and also the lecturer as a consequence of uncoordinated distribution. To prevent such issues, distribution is secured by two essential principles explained below.

Deployment of intermediate proposals

Every deployment has to go through a review process before it can be distributed. For this, deploying a particular lecture will automatically create a *proposal* in the Administration app as shown in Figure 2.3.4. Each proposal

Figure 2.3.4: Courseware proposals



is listed with a time stamp and the name of the author who deployed it. This allows users to maintain an overview of the different versions of a particular lecture and makes the distribution process fully transparent.

At the end of the deployment a preview of the online version of the courseware is opened (or available via the Administration app) for authors to review the content and formatting immediately. On the one hand, this review helps to reduce human errors—for example, finding typos or mistakes in formatting [3]. On the other hand, it establishes a workflow in which every proposal needs to be accepted (or rejected) to be distributed. Permissions can either be set so that only the lecturer, or that some (or all) of the collaborators, can approve proposals. Depending on the number and hierarchical setup of collaborators this enables different use cases without compromise with regard to convenience or control mechanisms.

Only one proposal at a time is accepted

By imposing a review workflow we make sure that no proposal get distributed without being accepted. In addition each proposal has a state indicated by the colored dots in the proposal list in Figure 2.3.4.

The yellow state, or *pending review*, is the default state for every deployed proposal. It either means that the proposal has not been reviewed or that a decision has not been made yet. Again, those proposals will never be visible to the students. The green state, *accepted*, is attained after a proposal has been confirmed for distribution. Depending on the schedule (discussed in the next section, 2.4), only these proposals are accessible to the students.[4] Finally, the red state, or *rejected*, is assigned to proposals that have either been rejected or have been replaced by another (green) proposal.

[3] Even for the case of only one author being the lecturer at any given time (i.e. no collaboration) the required review process cannot be disabled.

[4] Note that *accepted* does not mean *published*. The approval of a proposal is a necessary condition for distribution but not sufficient for it since lectures can still remain unpublished according to the schedule. The reason for this is to make the preparation of entire courses independent of the distribution schedule.

At no time there can be more than one green proposal. Once a proposal gets accepted while another proposal is in the green state already, the latter will automatically change to the rejected (red) state. This precludes conflicting versions of the same lecture.

At any time there can be several yellow or red proposals. Proposals don't get deleted automatically but users can delete them. A proposal that has been rejected or replaced can be reactivated by accepting it again (which will replace the currently accepted proposal). This allows educators to switch between different versions of a lecture easily should an allegedly newer or better version turn out to be flawed.

2.4 Scheduling

Proposals that have been accepted are distributed according to the lecture schedule. Instead of making materials immediately available, the schedule allows the precise timing of the distribution. This is necessary when courses are prepared in advance and the educators want to create a sequential learning journey for the students.

In SYLVA all course components follow the same scheduling logic. For any lecture or assessment educators can set independent publish, start, and end dates.

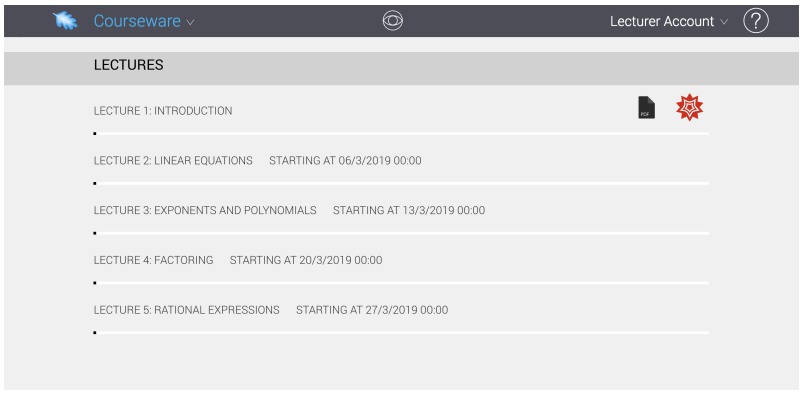
2.4.1 Publish, start, and end dates

When a new course is created there are no visible components to be displayed to students. If the course hasn't started yet but the lecturer wants to give an overview of the lectures and assessments contained in it, the components can be displayed (or published) without making the actual content accessible—the content does not even need to exist.

The *publish date* of a course component therefore determines when the component is shown to be extant and scheduled (like a preview). In Figure 2.4.1 we see an example of a course where lectures are scheduled on a weekly basis. The first lecture (here "Introduction") is already accessible to students and for file download, as indicated by the PDF and Wolfram Mathematica icons on the right side of the screen. The other lectures are shown to start at given dates and are not accessible (clickable) yet.

The *start date* determines when a course component is accessible to students. Consequently, the start date cannot be before the publish date as components that are not visible cannot be accessed. Finally, the *end date* limits the accessibility of the components with regard to their expiration. While such components will no longer be downloadable or accessible, they remain published, but are shown as expired. Again, for consistency the end date must not be before the start (and publish) date.

Figure 2.4.1: Courseware: list of scheduled lectures



These essential scheduling options also apply to the course level; that is, entire courses can be flexibly shown or hidden. Equipped with these features, educators can schedule every aspect of their courses in the initial setup. In the next section we will discuss how the schedule can be modified as the course is conducted.

2.4.2 Updating materials and dates

As previously mentioned, lectures can be updated (or replaced) by deploying or accepting new proposals. Similarly the schedule can be updated while the course is in preparation or proceeding, or when it has finished. However, to protect users from unwanted consequences some restrictions apply. To take a systematic look at the different cases, we distinguish four dates; the *current date (CD)*, the *publish date (PD)*, the *start date (SD)*, and the *end date (ED)*. With regard to the last three, the current date is either before or after it, representing two possible states. At any time we may want to *antedate* or *extend* the date, leading to two possible actions. Table 2.4.1 shows the options and restrictions, which can be summarized as:

- Antedating to either the current date or the previous date in the schedule is always possible *before* a particular schedule date has passed.

- Antedating a schedule date further back than the current date would not have any practical effect but could be misleading as it suggests (ex post) that a lecture has been published, started, or ended earlier. It is therefore disabled.

- Extending any schedule date that was scheduled to be before the current date is always possible up to the next schedule date, or without a time limit (for the end date).

Table 2.4.1: Courseware schedule modifications

Action	*before CD*	*after CD*
Publish Date (PD)		
antedate	up to CD	–
extend	up to SD	up to SD
Start Date (SD)		
antedate	up to PD/CD	–
extend	up to ED	up to ED
End Date (ED)		
antedate	up to SD/CD	–
extend	without limit	without limit

- Extending the publish or start date *after* the current date has passed effectively un-publishes or deactivates a lecture after it has already been visible or accessible. This action may violate some fairness or institution rules as some students may have seen the materials while others have not. This action should only be used to prevent greater harm and is protected from misuse by an additional warning.
- Extending the end date effectively extends accessibility before or after that date. As this action does not violate any fairness norm it is permitted without limitations.

Chapter 3

Assessment and Evaluation

3.1 Requirements for computer-based assessment

Using computers to take tests brings with it some of the most difficult technical and conceptual challenges. Assessments are usually high stakes situations: students are under pressure and trying to pass tests in the best possible way while educators take all the reputational and operational risks.

Computer-based assessments can contribute to improving the testing process and outcomes when they are fair, correct, robust, secure, and convenient to create and take (Smith, 2007; Gikandi et al., 2011). In the following sections we will discuss the requirements and limitations of this process in detail.

3.1.1 Fairness and transparency

In the context of assessments, several, partially overlapping concepts of fairness and justice are discussed in the literature, as summarized by Kunnan (2013). Within the scope of this thesis we will only discuss philosophical aspects from the perspective of whether they can be addressed by a computer-based assessment system, and refrain from more abstract and ethical considerations, such as distributive, retributive, or compensatory justice.

The use of technology in the assessment and evaluation process mainly affects the way tests are taken—that is, how, where, and when students access them, how questions are answered, and how the evaluation results are communicated. In this regard, providing equal opportunities to all students must be the goal if fairness is to be achieved.

In an indirect manner the design of the system also affects the way assessments are created and evaluated. Here the role of the technology will rather be to prevent discrimination, as decisions regarding content are made by the educator.

Providing equal opportunities

The most obvious requirement for providing equal opportunities to testees is the access to and timing of an assessment—that is, every student should obtain access to the same information at the same time and be allowed to work on answering the questions for the exact same time. In traditional (paper-based) assessments this is a major logistical challenge, in particular at the beginning and end of an exam. With limited staff it is almost impossible to distribute exams papers simultaneously to hundreds of students. Instead, exam papers are usually distributed in advance to the tables. This may allow some students to see the questions earlier than others—effectively giving them an advantage. To prevent this, staff need to monitor that all students start and finish the exam at the exact same time. At the end of the exam, in addition, staff need to collect the papers, which creates opportunities for some students to get some extra working time. While this may possibly give students only a couple more minutes it imposes a lot of stress on the staff for two reasons: First, if some students observe other students obtaining an advantage (even if this is only due to the staff's inability to monitor and enforce the rules of the exam), they can file complaints, which could lead to a repetition of the entire exam or to other legal consequences. Second, even when staff observe violations of the rules by individual students, the enforcement of the rules by means of disqualification from the exam is difficult. Given the high stakes situation for the student, clear evidence must be provided or the student can dispute the sanctions. Typically, this will involve additional paperwork for the staff and impose further risks since the evidence is, in most cases, based on (subjective) observations. Consequently, in the case of a lawsuit "he said—she said" situations with unpredictable outcomes arise. Beside these risks and additional paperwork, it will always remain questionable whether or when such interventions are appropriate given the negative consequences for the student.

One solution to improve the objectivity of the process is to move the enforcement of rules away from the staff. In this sense the assessment system acts as a third-party authority executing access and the timing of tests. With high levels of standardization and automation a computer-based assessment outperforms humans in terms of distribution precision and of speed, as well as of its ability to protocol all events on an individual student level. Similar to courseware scheduling, discussed in Section 2.4, access to assessments should be precisely timed. Then, computer-based assessment is likely to improve fairness compared to traditional assessment. In addition, to the benefit of both students and staff, it reduces noise and chaotic situations in larger classes, problematic cases, unnecessary human interactions, and waiting times.

When it comes to the content of an assessment—that is, the instructions, the order, and the formulation of questions—at first glance providing identical assessments to every student seems to yield the highest level of equality and therefore fairness. Thanks to distribution and grading efforts, in tradi-

tional assessments this is common practice. The same approach can be used for computer-based assessments too. With these, however, additional opportunities for the individualization of assessments by means of randomization and parametrization arise. These features can help to prevent cheating by distributing different versions of the assessment—that is, by making it harder to share (correct) answers among students (Arnold, 2016). One can argue that collusion and collaboration among students puts those who are solving the assessments on their own at a disadvantage. Consequently, individualized assessments would contribute to the equal opportunities precept. On the other hand, even slightly different versions of the assessment could lead to a variation in assessment outcomes that is not related to the student's learning success (Tversky and Kahneman, 1981). This trade-off cannot be resolved by the assessment system. Instead, the system should be designed to offer educators the choice to use these features according to their conception of fairness within the framework of academic and institutional standards.

Preventing discrimination

With regard to reducing or preventing discrimination, computer-based assessment systems can contribute mainly in the following three areas:

First, in many assessments success is correlated to the language skills of the student (Becker and Johnston, 1999; Favreau and Segalowitz, 1982; Cassels and Johnstone, 1984). Typically, foreign students perform worse in multiple choice assessments since they either need more time to translate the questions or do not understand the meaning of the questions and statements in the same way as do native speakers. On the one hand, technology allows us to make translations easier or at least faster. On the other hand, it can help to reduce the use of multiple choice questions by increasing the convenience of creating alternative, potentially less discriminating questions that assess the same learning objectives.

Second, to avoid disadvantages caused by the use of specific devices or operating systems, assessment solutions should allow for *bring your own device (BYOD)*. This has several practical advantages for conducting the assessment, but most of all it lets students pick the devices that fit their specific needs best and makes them feel more comfortable. However, one may then argue that the chances of succeeding in an assessment may dependent on the student's ability to afford specific devices, such as calculators, powerful laptops, or specific software for instance. Thus, with a BYOD policy a standardization of software tools and a centralization of computational infrastructure is required to attain fairness.

Third, automated grading is anonymous grading. Any type of social, personal, or individual preferences of human graders can be precluded.

Truthfulness

One of the maxims of assessment is to grade correctly—that is, a true answer should be evaluated as a right answer. Technology can help to automate the evaluation process and the calculations it requires. Therefore, it can reduce human error in the grading process. Technology cannot, however, guarantee truthful grading per se. As we will see in Section 3.3, one of the main challenges in assessing students by asking questions is to define and communicate the context such that a question is clear enough and understood by the student. This is not part of the grading process but is required for successful assessment.

From a grading perspective, in most cases the educator will provide components that are used to compute a solution, which does not need to be the truth. Even modern natural language processing technologies are not capable of correcting incorrect solutions. Therefore, a computer-based assessment system needs to allow ex post regrading and error recovery features.

In some cases it is possible to evaluate questions without the need to provide explicit solutions in advance. For this, verification conditions are computed that substitute the comparison of answers and solutions. While we cannot prevent human error, using automation can make it easier to detect, communicate, and resolve it. This must be seen as the highest potential avenue for technology to contribute to truthful assessment.

Transparency

Information asymmetry with regard to assessment and evaluation can lead to misunderstandings, require significant support efforts from educators, and even lead to distorted learning efforts or outcomes. In the most extreme case, students would only get a simple, single grade after completing a course, without further explanation, leaving them puzzled with regard to learning success, improvement potential, and their mistakes. In anticipation, students have incentives to spend effort on acquiring additional information about teachers and their preferences—instead of focusing on learning objectives and contents. In addition, a lack of transparency might lead to incentives to negotiate the assessment results with the educator afterward.

In many cases a lack of transparency is caused not on purpose, but by significant efforts made to create and distribute customized feedback (Gaytan and McEwen, 2007). Another cause of related problems is the lack of clear assessment objectives and grading rules. While this does not require customization on the student level, educators might be tempted to provide vague instructions as they can still derive the evaluation criteria once they received all the answers.

As a consequence, any ambiguity regarding the assessments will lead to student requests, exam or paper reviews, and additional office hours appointments. Beside the efforts of educators, students will also need to spend significant time on these things—time that could be better spent on actual learning.

The timing of the distribution of evaluations has some important implications too. Once the evaluation results are handed out, it is tedious to make ex post modifications because several students can be affected. This can, potentially, happen frequently—whenever, in fact, a student finds a mistake in the grading or an alternative solution to, or interpretation of, a particular question. Beside the logistical effort of collecting and correcting evaluations in such cases, educators might not even be aware of how many students are affected unless they keep copies of all student answers.

In conclusion, for any teacher–student relationship there should be the same information with regard to the instructions, rules, and evaluation criteria of an assessment. With respect to evaluation, all answers and evaluation results, and any modifications, should be tracked and made available to students and educators.

3.1.2 Robustness, security, and privacy

As the observation that assessments and evaluations need to be robust and secure while maintaining students' privacy is entirely uncontroversial, here we will only summarize the different perspectives on these aforementioned precepts.

Assessment privacy

Assessments, and in particular the answers students provide, must be treated as sensitive information since they, potentially, reveal a lot about the performance and various personal and psychological characteristics of a student. Such information can be used by companies in the context of an application, by banks or insurance companies for determining risk profiles, and even by governments for surveillance purposes. Therefore, any assessment must be private to the student in the sense that it is correctly authorized, individually taken, and all information about it, including all evaluations of it, must be kept private—accessible only to authorized staff and the student him or herself. This precludes, for instance, public APIs or data storage.

Assessment security

An assessment platform, first of all, must be secure against attempts at unauthorized access, including the access to any component such as a single question. Answers given by students and all related data, in particular evaluations, must be securely transmitted—that is, encrypted. All data that is generated, before or during the entire process of assessment and evaluation, must be secured by appropriate recovery and backup features. Since assessment-related data can become evidence in cases of disputes, it must be storable and archivable.

Academic integrity

To preserve academic integrity, beside that assessments must be taken individually, or by authorized groups only, two main challenges arise: First, no assessment or questions should be available before the assessment takes place. Second, answers to an assessment should not be easily sharable among students. To address both these goals, assessment individualization and randomization can be used. With regard to the first challenge, this implies that technology can restrict access and schedule the individualization such that related data does not exist until the assessment starts, and, therefore, cannot be accessed by anyone. With regard to the second goal, it implies that cheating—while not entirely precluded—can be discouraged when questions differ among students such that answers must differ too.

Robustness

From a technical perspective, an online platform provides a high level of availability since users can access it from almost any device, using a range of software, and from most places in the world. However, with the ubiquity of the Internet, the need for making various technologies compatible and maintaining that compatibility has become a huge technical challenge. In one of the first pilots of SYLVA, an educator set up an assessment with a duration of only three minutes. By then all kinds of mobile devices were supported for taking assessments. Some students used their smartphones to take assessments while travelling, and therefore experienced connection interruptions of several minutes in some cases. By the time they could reconnect the maximum duration of the assessment was exhausted and for fairness reasons no further attempt was allowed. This incident demonstrates that robustness must be thought through in a much broader context, but also that ongoing efforts will be required, not only to take into account technical specifications, but also to adapt to user behavior.

3.1.3 Usability

Beside increasing competition in the academic job market, educators are usually facing at least one of the following challenges with regard to assessments: increasing efficiency by assessing more students, in total, or more often; delivering more evaluations of students' learning, proving and improving students' progress.

To tackle these challenges without compromising on the quality of education (and without increasing the workload of educators), making the assessment process more convenient for educators is the most promising approach.

Automated grading and reporting

Automated grading can greatly reduce the human labor required for the assessment process (Baleni, 2015). The often tedious task of manually reviewing each student's answers not only takes a lot of time, it demands high attentiveness to avoid grading mistakes. The use of technology for grading therefore brings with it the highest potential for freeing up the time of educators.

Automated reporting can not only contribute to disciplining educators to provide clear instructions, it will also help to deliver evaluation results more rapidly to the student. Particularly in larger courses, grading can take so long that the questions that made up the assessment are no longer present in the students' minds. For effective learning, however, it is important that students can relate evaluation results to their actions, the effort they put into the course, and the answers they gave (Chen et al., 2010; Huba and Freed, 2000). This is even more important when students fail an exam and have to prepare for the retake.

Standardization

Using technology usually involves a learning and adaption process that requires additional time from educators and students. Due to the high potential for standardization, this initial investment can be overcompensated. First, the assessment process is highly repetitive and learning resources can therefore be shared among many users. Second, flexible templates for creating assessments can help educators to focus on assessing their subjects in the best possible way.

Collaboration

Similar to courseware, educators should be able collaborate on authoring assessments. In addition, collaboration in all subsequent processes—from the conduct of assessments to the evaluation of results, and handling of requests—has the potential to save educators significant amounts of time.

Tracking and documentation

One of the hardest tasks for educators is to keep track of (larger numbers of) students, and all their assessments, during a course. An online gradebook—in particular one that receives grades automatically from evaluated assessments—is of great value to any educator. While documenting is a tedious task per se, it becomes even more of a hassle for educators as bureaucratic standards increase and students get more emancipated in pursuing their rights—if necessary by means of lawsuits. While evidence-based decision making and the liberalization of students will not to be criticized here, it is important to note that all the burden of tracking is entirely on the educator.

3.2 Assessment properties, options, and types

In this section we will discuss different types of assessments and how they can be used for computer-based education in SYLVA. Conceptually an assessment is a framework composed of a set of properties, options, and questions to allow students to provide answers in a structured and consistent way. We will focus on the first two aspects here and discuss the actual assessment questions in more detail in the subsequent sections.

SYLVA is capable of the most common types of assessment for higher education—that is, self-tests, assignments, exams, and presentations. From a student's perspective the different assessment types may be associated with fundamentally different challenges; from a technical perspective, however, they are surprisingly similar. Table 3.2.1 gives an overview comparison of the four assessment types with regard to their properties and options.

As a general design decision, SYLVA is built to minimize the number of assessment types while maximizing flexibility with regard to configuration options. This allows educators to use the platform in the freest way and enables new or unconventional types of assessments, such as "team exams". This, however, requires educators to carefully configure the assessments, as some combinations may violate rules or laws imposed by their educational institutions.

We will start by discussing properties and options, before moving to specific use cases and characteristics of the assessment types, individually.

3.2.1 Assessment properties

Each assessment has properties that determine provision and availability, as well as its evaluation. In contrast to options, these properties are fixed (some of them due to technical limitations of the current implementation in SYLVA).

Scheduling and proposal review

For an assessment to be distributed, there needs to be a schedule and an accepted proposal. The schedule, analogously to courseware, discussed in Section 2.4, determines when the student sees that the assessment exists (publish date), when the student can access it (start date), and when the assessment can no longer be accessed (end date).

To prevent student confusion and human error by teaching staff, the scheduled distribution will only be triggered once there is a proposal for the particular assessment. Since SYLVA offers flexible collaboration features (see Section 2.3) an *accepted* proposal is a necessary condition for the assessment schedule to be activated.

In order to guarantee fairness, a few additional limitations apply to the scheduling options discussed in Section 2.4. The following actions are disabled or limited:

Table 3.2.1: Assessment types overview

	Self-test	Assignment	Exam	Presentation
Properties				
Scheduling	•	•	•	•
Proposal review	•	•	•	—
Provision	on demand	on demand	on check-in	on demand
Grading trigger	student	educator	educator	educator
Automated grading	•	•	•	—
Manual grading	—	—	—	•
Automated reporting	•	•	•	•
Student requests	•	•	•	•
Administration Options				
Introduction	•	•	•	•
Rules	•	•	•	•
Team	•	•	•	•
Graded	•	•	•	•
Weight	•	•	•	•
Time-restricted	•	•	•	—
Passing threshold	•	•	•	•
Attempts	•	•	•	n/a
Authoring Options				
Parametrization	•	•	•	—
Randomization	•	•	•	—
Partial credits	•	•	•	•
Negative points	•	•	•	•
Show solution	•	•	•	n/a
Show explanation	•	•	•	•
Conditions	•	•	•	—
Scoring rules	•	•	•	—

- Proposal locking: An assessment proposal cannot be accepted, updated, or rejected *after* the start date has passed.
- Start date locking: The start date of an assessment cannot be pre- or backdated *after* the initial start date has passed.
- Extended deadline warning: The end date of an assessment should not be changed *after* the start date has passed. Educators attempting to do this will see an additional warning and are required to confirm this action.

Specimen provision

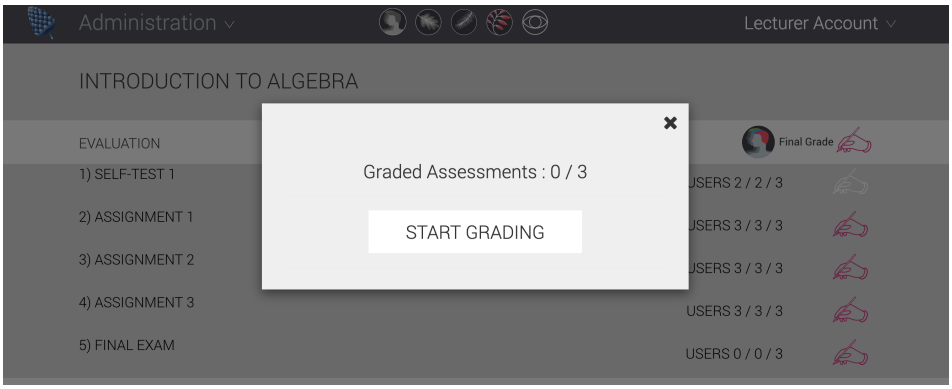
SYLVA provides assessment specimens—that is, individualized versions with specific parametrization and randomization—for each student based on a template the educator has created. To maximize the security of assessments, the process of providing the specimen should be prolonged for as long as possible. When the assessment specimen is only created when the student starts the assessment—that is to say, *on demand*—this combines with other technical security features to ensure that there is no way to access the assessment or know the questions in advance: the particular specimen does not exist until the student starts to work on it. Furthermore, this is also the most economical use of computational resources and storage since no specimen will be created when the student—for whatever reason—does not take the assessment.

The only downside of *on demand* specimen provision is the peak load in cases in which many students attempt to start the assessment at the same time, in particular when it is time-restricted. With the current implementation of SYLVA, this can lead to waiting times of up to a minute depending on the size of the assessment. To address this problem, the specimen creation is triggered by staff via the check in feature—for all exam type assessments. In large courses the authentication (check in) process usually takes several minutes—enough time to create all the specimens.

Grading trigger and automated reporting

All assessments in SYLVA have automated reporting, once the grading is finalized (for more details on reporting refer to Section 3.6). From a technical point of view there are almost no limitations with regard to the performance of grading. That means that even larger courses with hundreds of students can be graded instantly and automatically within seconds. For some cases however, this is not desirable. Whenever the educators want to prevent students from discussing or sharing solutions, for instance in a take-home assignment, the assessment should be graded only *after* all students have handed in their answers or when the deadline has passed in order to guarantee fairness. Another reason to defer grading is to allow educators to review students' answers and scoring and to adjust the grading if necessary.

Figure 3.2.1: Educator triggered grading



In SYLVA, therefore, two grading triggers are implemented: *Educator triggered* grading, as shown in Figure 3.2.1, gives teachers the flexibility to review the grading and report the results with an additional publish button. *Student triggered* grading, meanwhile, allows students instant feedback by displaying the results immediately after the submission of the assessment.

Automated and manual grading

The current implementation of SYLVA offers either *automated* or *manual* grading. Hybrid forms with some automatically and some manually graded questions are not possible. Automated grading is the default for all assessment types except presentations, which are usually graded based on several criteria, such as the performance and the answers to questions from the audience.

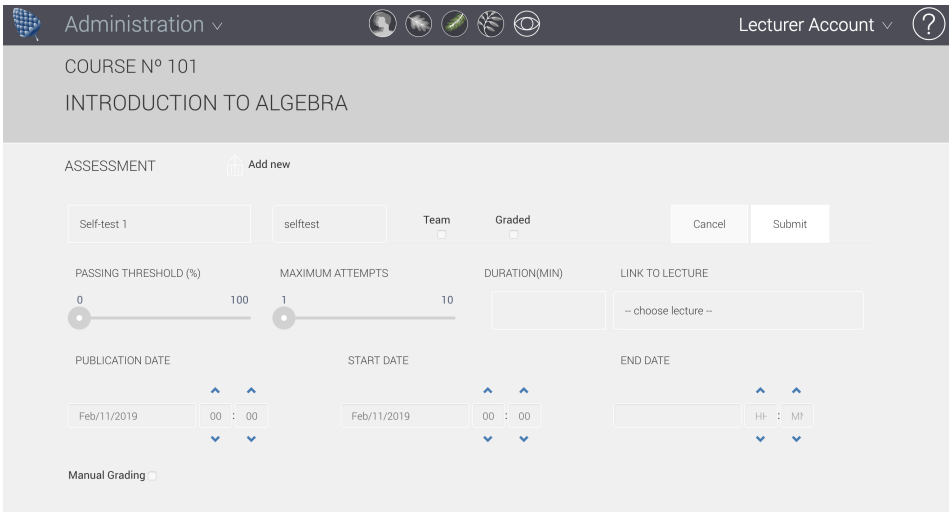
Student requests

Student requests allow students to ask for an adjustment of the grading of a particular assessment. Educators can handle these requests in SYLVA directly (rather than via separate e-mails) and in a structured and transparent way. The abovementioned features are discussed in more detail in Section 4.4. All assessment types in SYLVA have student requests by default.

3.2.2 Assessment options

Assessment options are parameters used to configure all aspects of an assessment that are not related to the actual question—that is, everything from the instructions through distribution to grading. As we can see in Figure 3.2.2, some of these options are set in the Administration app (*administration options* in Table 3.2.1), while others are set during the authoring process (*authoring options* in Table 3.2.1).

Figure 3.2.2: Assessment options UI



Introduction: welcome text and rules

For each assessment educators can provide specific introductions. In the welcome text some basic information along with motivational messages are provided. Rules are usually given as a list of allowed and banned tools, as are general instructions. Both are static text fields without further dependencies or functionality.

If the introduction is specified by the educators, it is shown once the assessment publish date is reached (for more details on scheduling refer to Section 2.4). This allows students to view the introduction before the actual assignment starts.

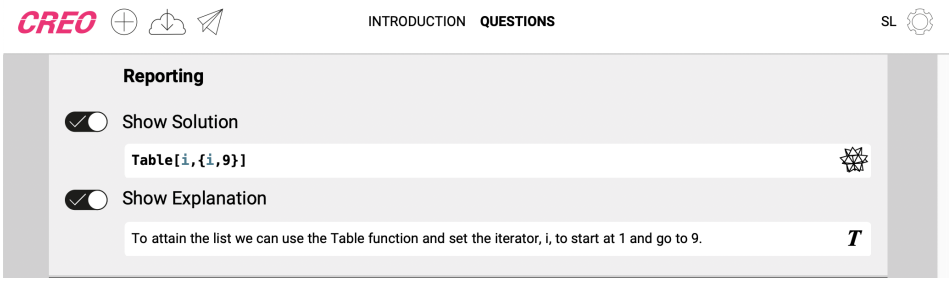
Team assessments

To foster collaboration and learning in teams, SYLVA offers team assessments, which can be answered in groups. For team assessments to be assigned, students have to be allocated to groups beforehand. More details on teams are discussed in Section 4.2. With regard to other options and properties, team assessments are identical to individual assessments (default option value)—that is, access to the assessment is shared between the members of the team; with regard, meanwhile, to the number of attempts and the grading, a team assessment is no different to an individual assessment.

Graded and weight

For each assessment the educators can choose if it counts toward the final grade of the course. Once the *graded* option is activated (default option value is

Figure 3.2.3: Show solution and explanation UI



deactivated), a *weight* can be assigned to calculate the final grade based on several assessments. Currently, there are no restrictions on the number format or interval of the weight as they are normalized for calculating final grades. For more details on grading refer to Section 3.5.

Show solution and explanation

When creating assessments educators can decide whether a *solution* and an additional *explanation* will be shown to the student in the report. Both options can be chosen at the question level, as shown in Figure 3.2.3. It is, for example, possible to show the solution only for some questions. Depending on the particular question the solution can be the only solution, one out of several possible solutions, or even a general solution that covers all cases. The explanation can be used to provide further information to facilitate an understanding of the solution, for instance by giving the solution path, or the grading by explaining the allocation of points.

Both fields can contain static text or expressions (including parametrization). For some cases of distinct questions the solution will be automatically derived. It cannot then be edited by the user, but can still be hidden from the report. Note that neither the solution field nor the explanation field are used for the actual (automated) grading. Both are independent and are for reporting purposes only.

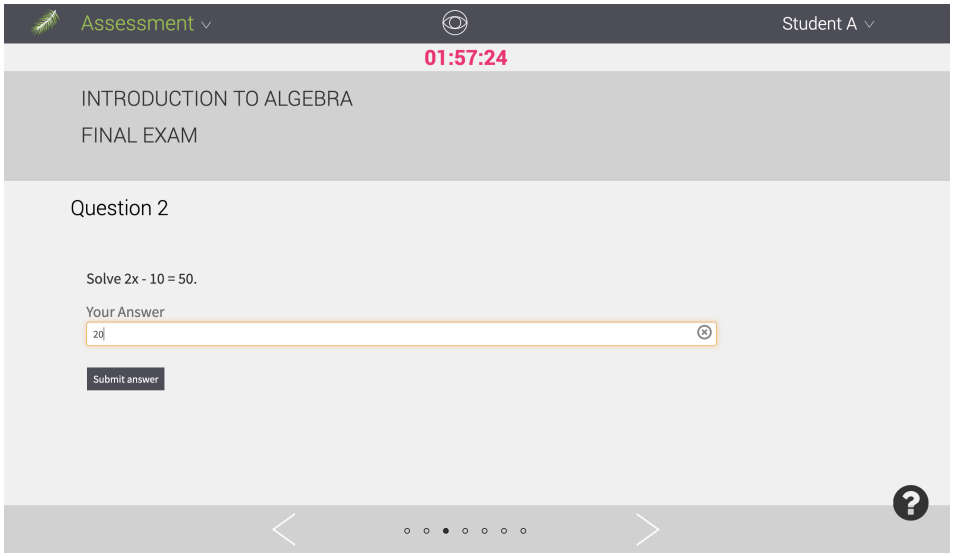
Providing a solution and further explanations in the student's report increases transparency and can therefore reduce the number of requests or questions. However, solutions revealed by the educators can be shared among students and are possibly known to students when the questions are reused.

Time-restricted

We already discussed the scheduling options for assessments in SYLVA in Section 2.4 and some special restrictions with regard to *scheduling and proposal review* above. As an additional option educators can set a duration for assessments—the time the students are allowed to work on the assessment once they have

started it. The duration, given in minutes, therefore cannot exceed the difference between end date and start date. It is a time frame within the two dates.

Figure 3.2.4: Assessment timer



This option is useful when educators intend to use time pressure as an additional challenge for the assessment. Setting a duration can also increase the fairness of the assessment. Once the duration is set, the students will see a countdown timer (see Figure 3.2.4) after they start the assessment to help them to stay on time.

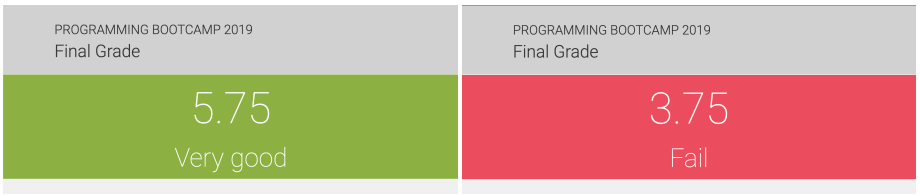
Passing threshold

The passing threshold is the minimum performance, in percent, which is required to *pass* the assessment. As shown in Figure 3.2.5, students with a performance below the threshold get a report that states that they *failed* the assessment, and vice versa.

In the current implementation of SYLVA the passing threshold only affects the reporting. There are no further consequences of (not) passing a particular assessment. The effective passing threshold with regard to the course is set by the minimum performance to pass the course in the grading scheme. For more details refer to Section 3.5.8.

Attempts

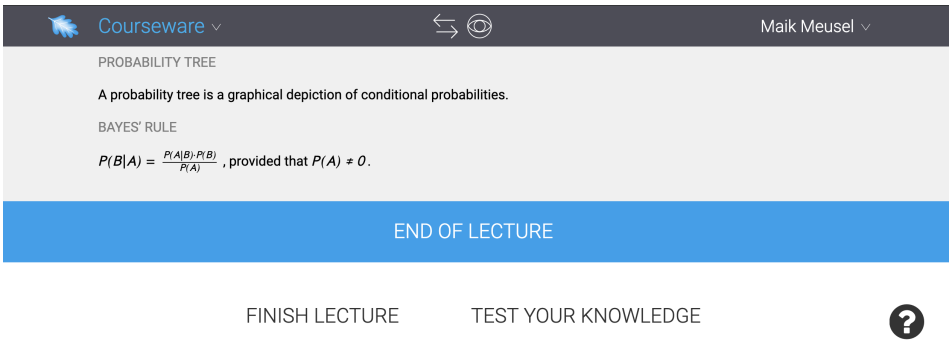
At the end of each assessment there is a button to submit the assessment. Since all answers given by the student are saved immediately after they click



(a) Passed (b) Failed

Figure 3.2.5: Passing threshold: failed vs. passed

Figure 3.2.6: Link to lecture



on "submit" in each question, the final submission of the assessment is not necessary to log in the answers.

For assessments where grading is triggered by the student, only the submission of the assessment will initialize the grading. Each submission counts as an attempt. The option *attempts* allows students to take the same assessment again—so, to start a new version (with a different parametrization or randomization). When the grading is triggered by the educator a new attempt will overwrite the previous submission of the assessment. Currently, one attempt is the default setting and up to ten attempts can be allowed.

Link to lecture

To facilitate a seamless learning experience, assessments can be integrated into the courseware via a button at the end as shown in Figure 3.2.6. The *link to lecture* allows educators to link each assessment to a lecture (chosen from a list of all existing lectures in the course). The assessments will still be listed in the assessment app, but especially for blended learning approaches linking the assessments to lectures is more convenient for the student and helps to structure the course.

3.2.3 Assessment types

In this section we will discuss how these options can be used to configure the assessment types. The examples given are based on assessment approaches commonly used at the University of Zurich and should therefore—at most—be seen as best practices.

Self-test

Self-tests are designed to help students to assess their knowledge and skills continuously during the semester. They are usually not graded and therefore comparable to the exercise question section in textbooks. In contrast to books, self-tests deliver an interactive and more convenient experience as the grading results can be seen immediately after the student submits the answers. With *attempts*, *randomization*, and *parametrization* students can repeat a particular self-test several times—with slightly different versions of the questions—to practice their understanding, or prepare for exams. For this reason, the solutions and explanations should not be shown in the evaluation report—otherwise students would see the solutions right after their first submission.

Assignment

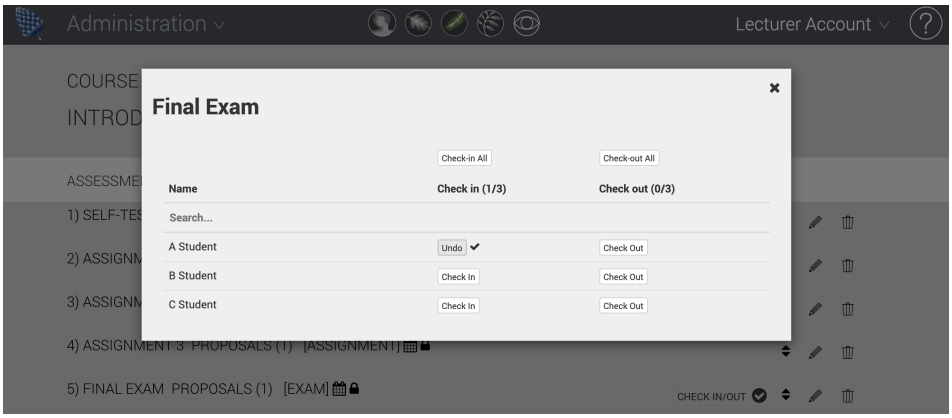
Assignments are usually graded assessments that mark milestones with regard to the learning objectives of a course during the semester. They have the potential to reinforce learning concepts by means of continuous assessment. Typically, assignments are given to students as homework that is solved either in groups or individually. They are usually scheduled such that students have several days to hand in their answers. Especially in the early stages of a course, educators are faced with higher performance heterogeneity among students due to different levels of prior knowledge regarding the topic. Therefore, assignments are usually not time-restricted to allow weaker students to catch up on the topics by applying extra (time) effort. To engage the best students, bonus points can be awarded for more difficult questions.

Exam

Exams are usually the final assessment—covering all learning objectives—of a course, and therefore count most toward the final grade. In many study programs exams are required to be sat individually to ensure that only personal learning success is credited. While in homework assignments the solution process cannot be observed by the educator, exams are conducted in-class to prevent collaboration between students, and to enforce higher authentication standards. This requires a precisely circumscribed schedule, not least for logistical reasons. Exams therefore usually have a duration within a narrow time window between start and end date. By default, students have to be checked in (as shown in

Figure 3.2.7) by authorized staff to access their exams. To assure academic integrity, *Randomization*, and *Parametrization* should be used when setting up the questions.

Figure 3.2.7: Exam check-in



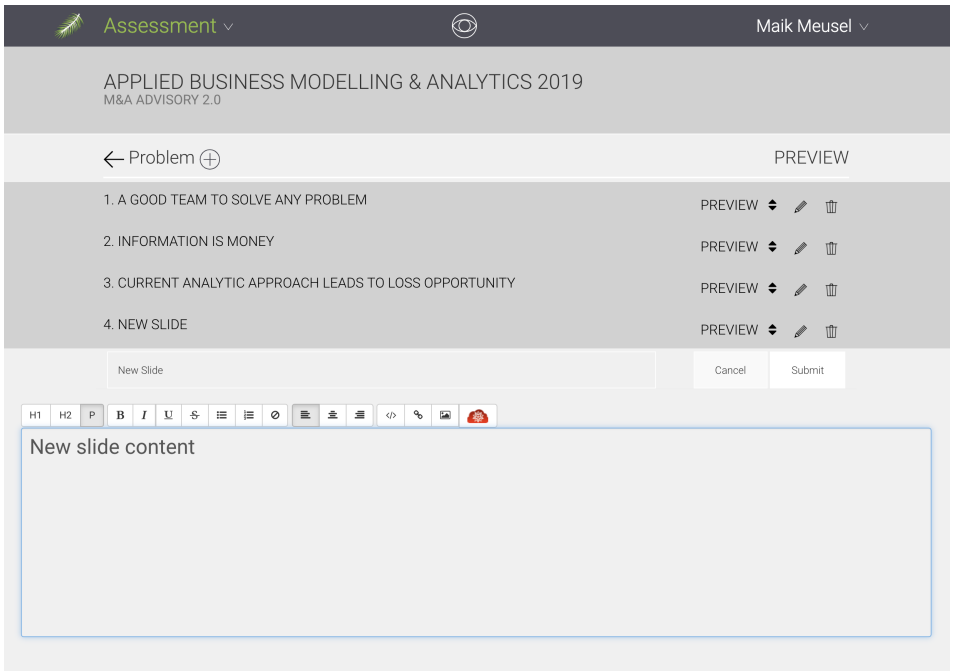
Presentation

In courses where students work on different projects, term papers or presentations are usually used for the assessment. Currently, these assessment types are not graded automatically—therefore we will put less focus on them in the following chapters.

SYLVA offers a web editor, where students can create slides with all elements that are implemented for the authoring, such as embedding static and interactive visualization, videos, and text. While the configuration of presentations is similar to that of the other assessment types, in Section 4.2 the setup of teams and projects is explained in more detail.

To edit slides students use the *Assessment* app, as shown in Figure 3.2.8. Presentations are also accessible from the *Courseware* app—as soon as the assessment has been started by the student(s)—in the form of the deployed version; that is, without editing features. Once the end date for the presentation has passed, students are no longer able to modify the presentation, but can view and use their presentations in-class as long as the course is accessible. This means that the scheduling of the assessment relates—consistent with the other assessment types—to the time students are allowed to work on it. Educators can monitor the students' progress in creating the presentations at any time.

Figure 3.2.8: Presentation editor



3.3 Question types

In this section we discuss different types of questions and derive examples to be used for selecting appropriate UI types, before we discuss the grading of those questions in the subsequent sections.

In the context of assessment we usually refer to questions, but as a more general notion we are dealing with tasks to be performed by students to test their knowledge, skills, or a combination of both. In this section the terms "questions" and "tasks" will therefore be used interchangeably. With regard to the goal of testing we will not distinguish between knowledge- or skill-targeted questions (or a combination of both) because it is irrelevant for the grading process and independent of the formulation of the question. To illustrate this, consider the question "What is the square root of 64?" The formulation of the question does not imply how the question is to be answered: some people may have learned the solution by heart (knowledge), while others derive it (possibly using a tool such as a calculator)—so, by using a certain skill. From this example it becomes clear that for most cases it is impossible to force a specific thought process, since only the outcome of this process (i.e., the answer) is observed. For the same reason the question formulation does not matter for the evaluation of the answer.

Therefore, we will focus on the types of questions with regard to their answer

space, or, in other words, discuss what can be graded automatically.

3.3.1 Distinct questions

An ideal situation for assessment are questions with a *distinct* solution. That is, a question that has only one unique and unambiguous solution, independent of any (external) circumstances such as time, location, language, etc. So, for example, questions regarding historical events such as "Who was the first president of the USA?" or "When was Albert Einstein born?" Distinct questions also include cases with analytic or derived solutions, such as "What is 3 times 4?" or "How long does it take to walk 10 miles at an average speed of 3 kilometers per hour?"

What all these examples have in common is the clarity of their solution. When there is only one correct solution, it is easy to compare this solution to any answer given by a student and verify or falsify it. Note, however, that the assumptions are quite restrictive and as a result such questions can be easily answered (correctly) by a computer with access to a web browser or services like Alexa[1] or Siri[2].

While distinct questions have desirable properties with regard to evaluating answers, their usage might be restricted to some forms of assessments, such as an oral exam (where the student is observed) or assessments with specific restrictions such as "no smartphone/no Internet" exams. As the availability, accessibility, and natural language processing capabilities of virtual assistants are increasing due to technological progress, it will become harder to use distinct questions for assessment of students' knowledge or skills.

Taking advantage of the convenience of distinct questions from an educator's perspective, a common strategy is to narrow down questions to distinct questions—so, constructing a case or example where all assumptions are stated to rule out any possible ambiguity. This allows one to ask more complex questions—addressing various skills and knowledge—without making the grading too hard. A typical example in higher education is a theoretical model, where parameters and assumptions are given and the student is asked to calculate a solution or a single parameter of this model. In high-school education such questions are usually called *word problems* or *math text problems*. Analytically they are more difficult than plain calculations because they demand processing things as units, definitions, parameters, and assumptions, which need to be combined, and involve some extent of reasoning. Currently, these questions cannot be answered using web browsers and virtual assistants as easily as can the examples given above. But this might change as natural language processing and computable knowledge progress.

[1] Alexa is an virtual assistant developed by Amazon (Amazon, 2019).

[2] The abovementioned examples have all been answered correctly by Siri. Siri is an artificial intelligence tool based on Machine Learning, designed to assist the users of apple products (Apple Inc., 2019).

3.3.2 Bounded questions

Bounded questions are problems that are no longer distinct—that is, have multiple possible solutions within an existing set of elements, such as intervals or groups. As a trivial example think of the following task: "Find an integer greater than 3 but less than 9." Obviously, this question has several correct answers: 4, 5, 6, 7, and 8. To grade a question like this automatically, it is no longer possible to simply compare *one* answer with *one* solution. Unless we state *all* possible solutions, in terms of grading this is a bigger challenge.

For assessment purposes bounded questions are valuable because they allow for higher levels of abstraction (and therefore difficulty). Instead of asking for a specific property of an instance, they allow one to ask for an instance (or instances) with specific properties. As an example, we could transform the question "Did Albert Einstein receive the Nobel Prize?" into "Name a physicist who won the Nobel Prize", or even "Name a physicist who won the Nobel Prize before 1930." Since we are not explicitly stating the instance, the focus of the assessment can be shifted to the properties, which is more interesting in many cases since it usually requires one to process more information and/or apply various skills. Note that search engines or virtual assistants are less likely to answer such questions correctly for the same reason: the requirements of logically understanding such tasks are higher since mapping of assumptions/properties to related instances is necessary and additional skills, such as calculations, sorting, etc., can be included. To illustrate this let's consider the following example: "Find a palindrome in the declaration of independence from July 4, 1776." Assume the learning objectives are to understand the concept of a palindrome and to be able to identify it in a random text. With a distinct question we could ask students for the definition of a palindrome, to select a palindrome from a list of words, or to find a palindrome in a text with exactly one palindrome. The first two options, although related to the learning objectives, do not address the skill of identifying the abstract concept of a palindrome in the context of a regular text. While the last option gets close to the assessment goal, still it is not possible to give the student a random (real) text. Instead a text needs to be prepared and once one student has found the palindrome it's relatively easy to share it with others (depending on the assessment type).

A practical problem in assessment, which can be addressed with bounded questions, is rounding and approximation. Particularly in quantitative fields the answers to questions are often numerical. Consider the simple question "How many buckets of paint do I need to paint a disk with a radius of 100 kilometers?" Let's assume the learning objective that motivated this task was to find out if the student is able to apply the abstract concept of calculating the area of a disk and how many buckets of paint are needed to fill it in a real-world context. Based on this question it is very unlikely that students, who know how to calculate the solution conceptually, will come up with a unique answer. First, the numerical solution depends on the precision of the constant Pi used

for the calculation of the area. Second, there are different types of paint, which may vary in terms of the area they can cover. Third, there are different sizes of paint buckets. Fourth, some students may assume that it is not possible to buy or use fractions of a bucket and therefore round their solutions up to full buckets—while others may assume the opposite. Within the framework of distinct questions, the educator needs to either narrow down the question to rule out any ambiguity, or review all the solutions (and assumptions made by the students) manually. Both alternatives are rather unattractive. The former would blow up the question text with all the necessary assumptions involving precise instructions on how to use the constant Pi, rounding rules, etc, which reveals parts of the initial challenge. The latter alternative leads to such a high grading effort that it becomes questionable if the initial learning objective can still be justified. Note that the problem discussed by way of this example applies to almost all questions with a real-world context. As a consequence questions in conventional assessments remain mostly abstract (unrelated to real problems), unintuitive (unnatural language; possibly misleading due to additional assumptions and rules), and synthetic (constructed for educational purposes).

To summarize, bounded questions can be used for many cases to lower the restrictive assumptions needed for questions to be distinct. This allows educators to assess skills (and related knowledge) in a more realistic, problem-oriented context. At the same time, bounded questions offer more flexibility, in particular to ask more complex or difficult questions, without compromising convenience in grading.

3.3.3 Open-ended questions

In contrast to distinct and bounded questions, open-ended tasks can be characterized as having no ex ante determinable or predictable solutions whatsoever. These questions, therefore, offer the highest flexibility with regard to abstraction or real-world context. Since no further assumptions are required *every* question is at least open-ended.

Obviously, from an assessment perspective being able to grade open-ended questions automatically is the ultimate objective. At this point one may ask how this could possibly be achieved given that the solutions cannot be determined in advance. Let's approach this seemingly paradoxical proposition by lowering the restrictions of bounded questions. As a first example, recall the following question from the previous section: "Find an integer greater than 3 but less than 9." If we remove the second constraint (bound), we already have an open-ended question: "Find an integer greater than 3." To grade this question automatically, it is no longer suitable (or even worthwhile) to try to list all possible solutions—there are infinitely many. Surprisingly, considering manual grading, this question is still easy to grade. The reason is that to grade this question one can apply conditions, such as the number must be positive and have no decimal places or fractions, to classify it as correct or wrong. The use of

abstract conditions allows one to grade an answer that may have been unknown ex ante. Therefore, to grade such questions automatically, we need to replicate the same idea—that is, to apply conditions that determine the correctness of an answer.

As a next step let's discuss a class of open-ended tasks that can be characterized as the "superfluity problem". Consider the following examples: 1) "Find an equation with one parameter that has the solution 5"; 2) "Write code that evaluates to the list $\{1, 2, 3, 4, 5\}$."; 3) "Define the concept of a 'steady state' in economics." What all these examples have in common—despite the fact that there is neither a distinct nor a bounded solution—is that we cannot make any reasonable assumptions about the sheer size of the answer. For the first example, it is feasible to use all kinds of combinations of mathematical operations and quantifiers. Analogously, for the second example the only limitations for an answer are given by the programming language. For the third example, one could use an illustrative example, or two, or three, etc. Still, exerting the idea of conditions, we can grade all the questions automatically. For the first example we can state the condition "When parameter is replaced by 5, the left hand side of the equation equals the right hand side of the equation"; for the second simply "The code evaluates to the list $\{1, 2, 3, 4, 5\}$"; and for the last "A steady-state economy is characterized by a constant capital stock and a constant population size." In other words, for any set of conditions that determine the correctness of an answer, we can formulate a minimal sufficient solution and add arbitrarily many additional elements or arguments. This has the great advantage that we do not need to restrict the answers—that is, force the student to adopt (anticipate) the format of the grading. Therefore, open-ended questions offer the most natural way of articulating answers.

3.3.4 Special cases and limitations

In the previous sections we discussed three classes of questions, their assumptions, as well as practical and grading-related implications. While all question types can be graded automatically, in this section we discuss conceptual and technical limitations.

Indeterminate questions

So far we have dealt with questions that have one, some, or infinitely many solutions. Herein the term *solution* is associated with a verifiable expression of a single entity, an instance or group of instances, or abstract conditions. Recall the third example from the last section: "Define the concept of a 'steady state' in economics." As a specific term from the field of economics, the definition might depend on the context (e.g., model), or time and scholar. As a result we can assume a variety of existing definitions, which overlap or even contradict each other to some extent. This is not a problem as long as we assume that

the educator either provides the context to rule out ambiguity or formulates the conditions in such a way that contradicting definitions can be distinguished. However, the automated grading has to be based on conditions that are verifiable.

Now consider the tasks "Draw a beautiful picture" or "Write a harmonic poem." The attributes *beautiful* and *harmonic* imply a certain level of ambiguity. Whether a picture is considered beautiful or not may in the end be a matter of taste. The only approach to grading such a question (assuming there is no definition or further context given) is then to formulate the conditions as exclusions—that is, attempts to falsify conditions.

In the extreme case of a question such as "What is your favorite color?" there is no meaningful way to formulate conditions that qualify a "solution". Therefore the question is indeterminate. Note that the capabilities of automated grading are neither superior nor inferior to those of humans carrying out grading. Within the boundaries of the laws of logic, all questions can be graded with a condition-based approach.

Ex post validation

Consider we are planning to ask the following question: "['Yes' or 'No'] Was it raining in Zurich on December 31 of this year?" As long as this day hasn't passed the question could possibly be answered by using a weather forecast (or by estimating the likelihood of rain), but once the day has passed there will be a distinct solution ('Yes' or 'No') to this question—that is, we only know the solution for sure from a certain point in time on. This is the special case of questions where the solution can be indeterminate at the time the question is created (or even when it's asked).

Now, if the question is answered before the last day of the year, it is not possible to grade the answer before this last day of the year. As soon as the day has passed, however, we can grade the answer by validating a condition, ex post. Thus, using conditions for the grading allows us to not have to know the solution (only that it exists) in advance. Automated grading is therefore not required to be predetermined.

The example discussed above is arguably not a commonly used type of questions; under certain circumstances one may even question if it makes sense to ask such a question at all—who can see into the future? Note, however, that we have the same flexibility in asking and grading questions regarding the future as humans do. This is in particular important for adaptive assessment, an assessment approach where the grading and sequence of questions depends on the answer to a previous question (which is unknown at the time the questions are created and asked).

Interdependent questions

Following up on the previous example where we discussed the grading of time-dependent questions, let's think of other dependencies. Consider the following task: "Pick a number between 0 and 100 that is closest to 2/3 of the average of all the numbers picked by the other students for this question." This example is based on a problem that is studied as the "guess 2/3 of the average" game in experimental economics and game theory (Nagel, 1995).

At this point we won't discuss if this question can be seen as fair and how grades would exactly be allocated. Note that this question—similar to the one discussed in the last section—has a distinct solution that is unknown at the time when the question is created and asked. Analogously, we can grade such a question automatically if we use conditions that are validated after all the students' answers are collected. Now, however, the solution depends on the answers given by *all* students—it is interdependent.

The idea of interdependent questions, again, entails potential use cases beyond single question applications. For the grading of a question the educator could formulate conditions based on other students' answers—for example, "The answer given is similar to the most common answer/the answers from students who answered other questions correctly." Theoretically, this could even be used as the only grading approach, resulting in a form of automated peer-grading.

3.4 Answer types

In the last section we discussed different question types and what it takes to grade them automatically. In this section we will focus on how these questions can be answered. By many tools such as Möbius (DigitalEd, 2019) question types are classified by how the students are supposed to answer them. This can create the impression that certain types of questions can only be answered in a particular way, or even that specific questions require specific answer types.

When you consider a trivial question like "What is 1 + 2?" it is easy to realize that are many ways to give an answer to this question: The student could be faced with a statement like "1 + 2 = 3" and be asked to make the decision whether the statement is true or false. It is also possible to present more than just one statement and let the student select the correct one among several alternatives. One could even think of presenting several correct and incorrect alternatives and asking the student to select all correct statements. We could ask the student to answer the question by typing in a number, or a word ("three"), to write code that evaluates to the solution, or to upload a spreadsheet or an image with the number. Finally, we could ask the student to answer the question interactively by moving a slider, pointing to a number or dragging and dropping elements; and there are many more possibilities.

As we can see, there are various ways to answer a particular question. Not

all of them are equally difficult from the student's perspective, but they are all equally suitable. Therefore it is up to the educators' creativity to specify how a question is supposed to be answered. The reason for classifying questions by answer type can be seen in the restrictions of user interface templates and differences with regard to grading effort. In SYLVA these limitations are no longer present as automated grading of all different answer types is effortless for the educator. This results in higher flexibility for educators.

The trivial example discussed above includes the three main classes of answer types that are implemented in SYLVA: *Selection*, *Free Response*, and *Interactive Response*. In the following sections we will discuss them in more detail and derive concrete question examples. The grading will then be explained in subsequent sections.

3.4.1 Selection

Giving an answer by selecting one or several options asks the student to make decisions. In contrast to other answer types, here the educator provides the correct choice concealed between other, incorrect choices. Since the option space is limited, the student can answer the question by finding the right choice, or by eliminating (all) the incorrect choices. Consider the example of a student who does not know the answer to a particular question, but can identify the options, which are wrong. Then, the student would be able to give a correct answer, but the question is not appropriate to assess the learning success. This implies two challenges for the educator: First, there must be at least one unambiguously correct choice, otherwise the question will be invalid. Second, if there are not enough sufficiently similar or credible alternative incorrect choices, the solution can be guessed easily. This can lead to a trade-off between the risk of not having enough choices and introducing ambiguity by adding choices that are too similar to one another.

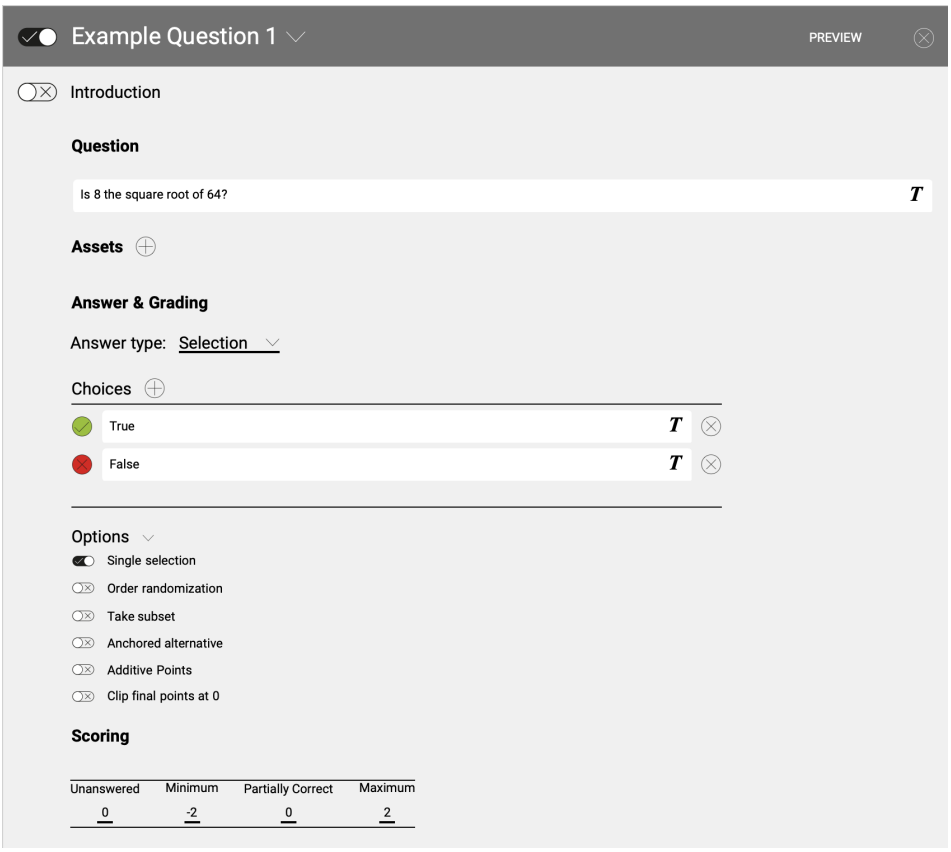
Example Question 1

A common use case of selection questions are questions that ask the student to either accept or reject a statement—true-or-false questions. As a first example let's consider the following question: "Is 8 the square root of 64?" Figure 3.4.1 shows how the question is setup in CREO and Table 3.4.1 summarizes all the relevant information to track the grading, including conditions and scoring rules, which are not explicitly visible in CREO.

As the statement in the question must be either true or false, it does not make sense to add more than the two choices or allow more than one choice to be selected. Therefore, the questions is set up as single selection, not taking a subset, and without an anchored alternative.

Since the two choices are logical opposites, there is no room for interpretation of the decision made—that is, there are no partially correct answers. Thus, the

Figure 3.4.1: CREO setup of Example Question 1



conditions are logical antagonisms too, which can never both be *True* at the same time. To specify the scoring, it is, then, most convenient to use *holistic* consolidation (*additive points* option turned off); that is, to assign points to the basic possible outcomes, rather than derive the scoring from the conditions separately.

As mentioned in the introduction to this section, for a question like this with not many choices students have a higher probability of finding the solution by random guessing. In this example the probability of randomly selecting the correct choice is obviously as high as 50%. Now, if we would award two points for correct answers and no points otherwise, then the expected value from random guessing would be one point, creating strong incentives for every student to answer the question—no matter if they know the answer. To prevent such incentives, we can set two negative points for wrong choices to neutralize the positive expected value from random guessing.

Note that negative points are risky. If the assessment would only consist of two such questions, answering one of them incorrectly means that the student's

Table 3.4.1: Example Question 1: True-or-False

Question	Is 8 the square root of 64?	
Options	Single selection ⇾ True, Additive points ⇾ False	
Choices		
choice1	"True"	
choice2	"False"	
Conditions		
condition1	choice1 is selected	
condition2	choice2 is selected	
Scoring Rules		Points
scoringRule1	condition1 is $True$	2
scoringRule2	condition2 is $True$	-2
scoringRule3	no answer was given	0

performance can not reach more than 0%. Therefore, by default points are clipped at zero. To enforce negative points the option needs to be turned off. Now that the incentives for random guessing are eliminated, we can set zero points for unanswered questions (to not punish the student for not knowing the answer, only for random guessing).

At this point, to keep the example as simple as possible, we will not use parametrization and randomization as this would only increase the complexity of the notation without generating additional insights.

Example Question 2

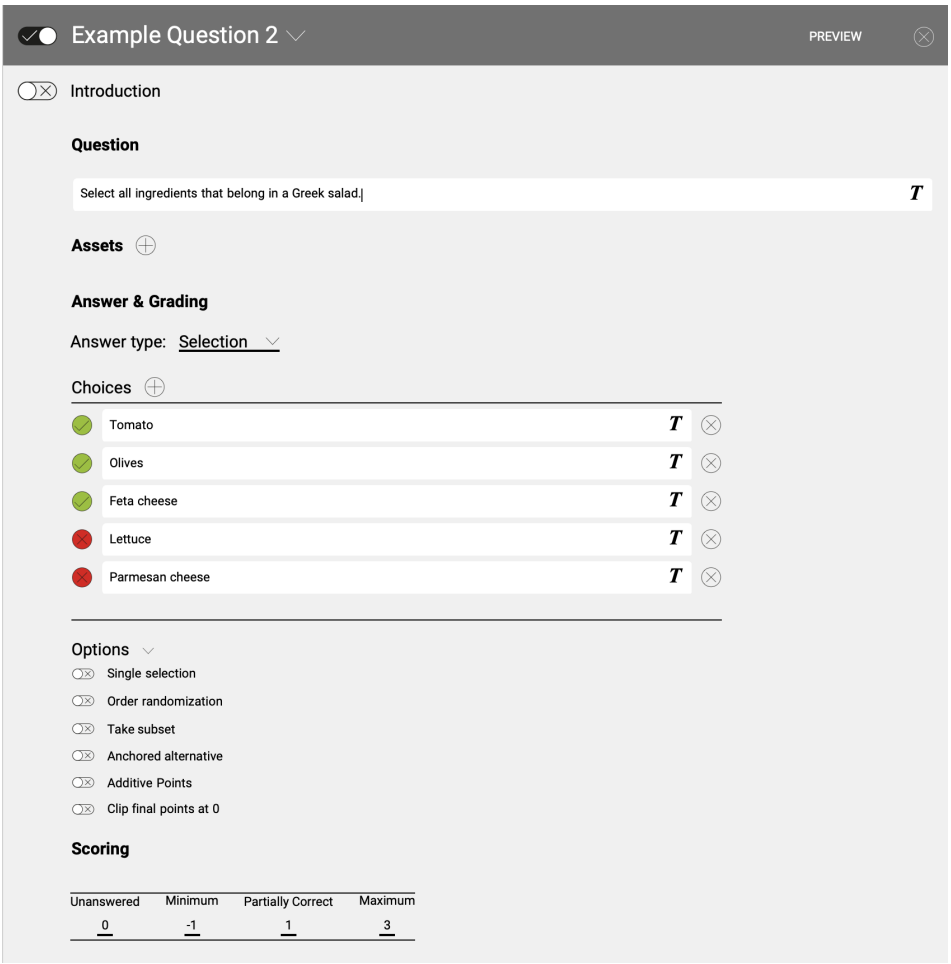
For the first example question we only allowed a single choice to be made by the student. Once we allow multiple selections, we can ask questions about sets or collections of statements, items, or properties.

Note that just like in the single selection case, making a particular choice also means not making the opposite choice. In a setup of multiple selections one question can be interpreted as asking for several decisions for each of the statements (from a set of statements).

For our second example question we ask: "Select all ingredients that belong in a Greek salad." The CREO setup of this question is illustrated in Figure 3.4.2 and all related information is summarized in Table 3.4.2.

While the setup of the second example question is similar to that of the first one, allowing for multiple selections (single selection option turned off) allows us to mark several choices as correct, denoted by the green checked disks next to the choices. In our example we have three ingredients—that is, tomato, olives, and Feta cheese—that belong in a Greek salad, while lettuce and Parmesan cheese rather belong in a Cesar salad.

Figure 3.4.2: CREO setup of Example Question 2



Now, when it comes to the scoring, we would assign the full three points to an answer where all the right and none of the wrong ingredients are selected, and zero points in the opposite case. But what if a student, for instance, only selected two of the right ingredients? This answer would be neither fully correct nor completely wrong. In this case we can award one point for partially correct answers.

With holistic consolidation all selections that are neither (fully) correct nor wrong are considered partially correct (as expressed in $scoringRule2$). At this point one may argue that we could (or should) further distinguish different cases of partially correct answers. A gastronome could claim that putting lettuce in a Greek salad would be still acceptable while Parmesan cheese would ruin the salad. On the other hand, some people would maybe not call it a Greek salad when Feta cheese and olives are missing.

Table 3.4.2: Example Question 2: Select all

Question	Select all ingredients that belong in a Greek salad.	
Options	Single selection ⇾ False, Additive points ⇾ False	
Choices		
choice1	"Tomato"	
choice2	"Olives"	
choice3	"Feta cheese"	
choice4	"Lettuce"	
choice5	"Parmesan cheese"	
Conditions		
condition1	choice1 is selected	
condition2	choice2 is selected	
condition3	choice3 is selected	
condition4	choice4 is not selected	
condition5	choice5 is not selected	
Scoring Rules		Points
scoringRule1	all conditions are $True$	3
scoringRule2	at least one condition is $True$	1
scoringRule3	none of the previous scoringRules is $True$	-1
scoringRule4	no answer was given	0

Note that all those distinctions could be achieved using additive consolidation. However, those ideas would impose an additional ranking with regard to the importance of each ingredient, which is not explicitly demanded in the initial statement of the question. Furthermore, setting further nuances in the grading requires not just more specifications in CREO but also more careful thought (tastes differ).

With regard to the transparency of the scoring, more complex rules also require more explanations. In this example, we see a trade-off between simplicity (convenience) and the preciseness of the grading. In the next example we will discuss how to setup a question with additive consolidation.

3.4.2 Free response

Whenever we don't want to provide choices via which to answer a question, we can use the *Free Response* answer type. As the name suggests, this type is designed to accept any kind of answer. In many situations, however, it is useful to make some restrictions to the type of answer that is expected to help the student to submit a valid answer, and to simplify the grading.

Consider a question asking for a numerical result of a calculation. In this case we would like to make sure that the student does not (accidentally) enter a character or try to upload a file. If the task was, however, to do the calculation

in an Excel spreadsheet and upload the file afterward, then we would like to prevent students from uploading a PDF file, for instance.

Once the answer has been submitted, it is necessary to match the format of the conditions to be graded. To give an example, when the question is to name the first digit of Pi—in the absence of further restrictions–students could answer "three", "3", or even "1+2". Instead of having to think about all possible inputs (and how to transform them into a matching format for the grading), it is more convenient to interpret the answer before the grading starts.

To achieve both features, Free Response questions have an additional selection field, the *Interpreter*, to specify the type of answer that is expected (or into which type the answer will be transformed). The Wolfram Language offers 689 different interpreter types[3], including various text, number, and file formats. In addition many types of natural language interpreters, for instance city, country, food, or insect, can be used. For special types of inputs such as dates or colors, specific UI types are automatically provided to the student— in these cases a date or color picker.

For assessment purposes not all of the interpreter types are useful. CREO therefore offers a selection of common types to be used for Free Response questions. This gives educators a wide variety of possibilities to ask all kinds of different questions.

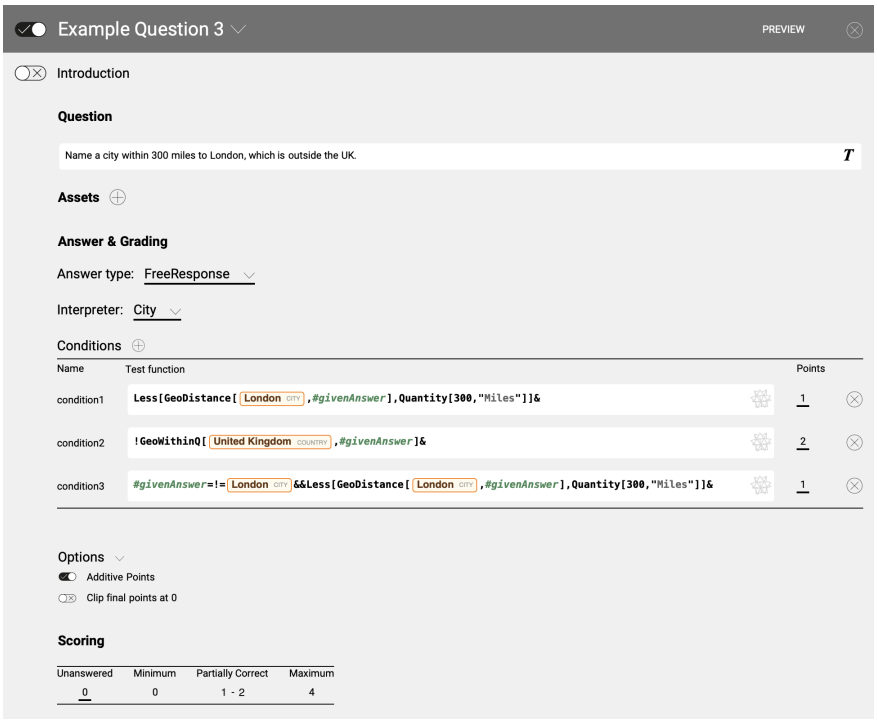
Example Question 3

For the next example we will take advantage of interpreting locations for the following Free Response question: "Name a city within 300 miles of London, but outside the UK." This is a bounded question since there are many cities that satisfy the required conditions. Instead of trying to provide a list of all solutions to the questions we can check for any given answer if the conditions hold. Whenever the conditions are logically independent it is convenient to use additive consolidation (additive points option turned on). A summary of the conditions and scoring rules is given in Table 3.4.3 and the CREO setup is shown in Figure 3.4.3.

For this example we can check the following conditions: (*condition*1) The distance from London to *givenAnswer* is less than 300 miles, and (*condition*2) the *givenAnswer* is not in the UK. This is possible when the *givenAnswer* is interpreted as a city (rather than as a string for instance). Checking the conditions independently allows us to distinguish answers that are clearly correct, or clearly wrong, but also those that are partially correct—that is to say, only satisfy one of the two conditions. In this example we might award half of the points when the *givenAnswer* satisfies the first or the second condition. Note that there are no restrictions with regard to the city that is accepted as an answer. In our example the answer "London" would be valid too. Obviously,

[3]Based on Mathematica version 11.3.0.0. An overview of all interpreter types is provided here: https://reference.wolfram.com/language/ref/$InterpreterTypes.html.

Figure 3.4.3: CREO setup of Example Question 3



this answer would satisfy *condition*1. Now, one may argue that we should not award points for submitting the city which is already mentioned in the question. While technically London is less than 300 miles from London, we can not be sure if the student understood the concept of calculating a distance, or if this is just a lucky coincidence.

Table 3.4.3: Example Question 3: Knowledge-based

Question	Name a city within 300 miles to London, but outside the UK.	
Options	Additive points ⇾ True, Interpreter ⇾ City	
Conditions		
condition1	distance from London to *givenAnswer* is less than 300 miles	
condition2	*givenAnswer* is not in the UK	
condition3	condition1 is *True* and *givenAnswer* is not London	
Scoring Rules		Points
scoringRule1	condition1 is *True*	1
scoringRule2	condition2 is *True*	2
scoringRule3	condition3 is *True*	1
scoringRule4	none of the previous scoringRules is *True*	0
scoringRule5	no answer was given	0

One way to address this problem is to split the initial condition. Instead of awarding two points for each of *condition*1 and *condition*2, we add a third

condition, which further restricts *condition*1 by explicitly excluding "London". Then, all answers that satisfy *condition*3 (one point) will also satisfy *condition*1 (one point). In this way we add another group of partially correct solutions to the grading. As shown in Table 3.4.3 the maximum number of points add up to four.

With additive consolidation, the scoring rules are automatically derived by CREO: For each condition one scoring rule associates points when the condition yields $True$. In addition one scoring rule associates points to the cases where none of the conditions are $True$ and one when the question is unanswered, respectively.

Example Question 4

In the next example we will discuss an open-ended question with Free Response. One of the many applications for such a question is a coding question. Consider the following example: "Write code with less than 20 characters that evaluates to the following list: $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$."

Grading code is challenging since there are many different ways to get to the same result. In this question the length of the code is required to be less than 20 characters. While it would be possible to set up the interpreter to only accept answers with 19 characters or less, this would require further customization of the interpreter beyond the default options given in CREO. One might argue that some students fail to come up with a code that satisfies both requirements but do get to the list at least. In that case their answer could be counted as partially correct.

To preserve the code that is submitted as an answer from being evaluated, in this case we can set the interpreter to *String*. Then we evaluate the code in the conditions. If we evaluated the code immediately, we would lose the initial sequence of characters in the *givenAnswer*.

Now, to state the conditions for the grading of this question it is important to keep in mind that this is an open-ended question. Even if we would restrict the answer to being 19 characters or less, there would still be more than 10^{40} possible different answers (assuming 128 ASCII character basis). The Wolfram Language has thousands of built-in functions that could be combined to answer this question.

Therefore, when it comes to coding questions, we should not attempt to base the grading on matching a particular solution or patterns such as the use of certain functions.

Even if we managed to list all the combinations of possible solutions, it would be computationally inefficient for each answer to search for a matching solution (in many cases that would be altogether impossible). Instead, we will focus on defining conditions that qualify correct answers, and exclude cases that qualify incorrect answers.

Especially for coding questions we are likely to end up with several conditions

Figure 3.4.4: CREO setup of Example Question 4

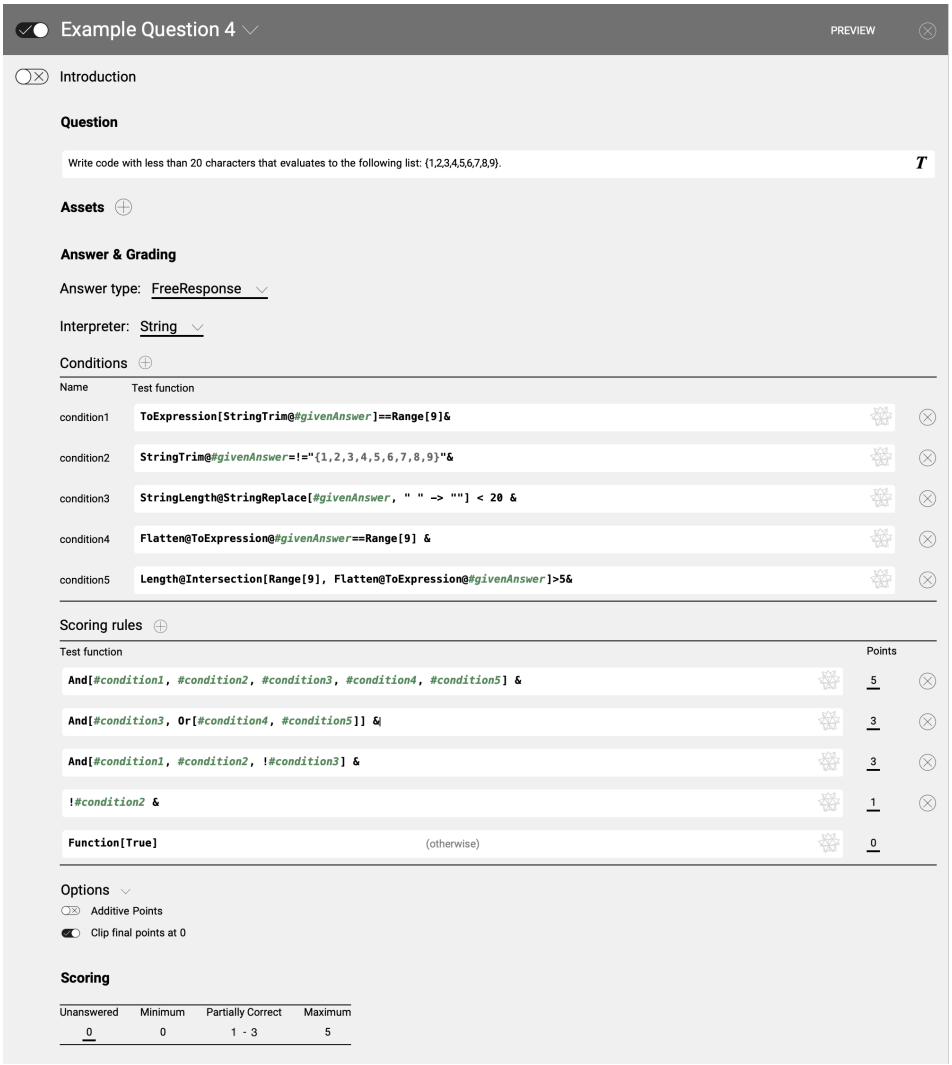


Table 3.4.4: Example Question 4: Coding

Question	Write code with less than 20 characters that evaluates to the following list: $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.	
Options	Additive points → False, Interpreter → String	
Conditions		
condition1	evaluated *givenAnswer* is equal to *Range*[9]	
condition2	*givenAnswer* is not literally $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$	
condition3	number of characters in *givenAnswer* is less than 20	
condition4	evaluated and flattened *givenAnswer* is equal to *Range*[9]	
condition5	evaluated flattened *givenAnswer* and $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ have more than 5 elements in common	
Scoring Rules		Points
scoringRule1	all conditions are $True$	5
scoringRule2	condition3 and conditions 4 or 5 are $True$	3
scoringRule3	conditions 1 and 2 but not condition3 are $True$	3
scoringRule4	condition2 is $False$	1
scoringRule5	none of the previous scoringRules is $True$	0
scoringRule6	no answer was given	0

that are not logically independent. Therefore, it is more convenient to use holistic consolidation for this example— that is, to list all the conditions first and define scoring rules afterward.

In *condition*1 we can state that *givenAnswer* evaluates to the same output as our solution—that is, the required list. Just like in the previous example, a trivial answer to the question would simply be to repeat the list. Evaluating the list yields the same list again—this answer would satisfy our first condition. To avoid this we can exclude this case explicitly in *condition*2.

Since we did not restrict the number of characters in the answer initially, we want to make sure that it does not exceed the limit in *condition*3. At this point we don't need further conditions to identify a fully correct answer. Now we can think about cases which we would consider as partially correct answers. One of the difficulties of working with lists in the Wolfram Language is the correct placement of curly brackets. Once we add a pair of brackets to each element, for instance with the code $Table[\{i\}, \{i, 9\}]$, we get $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}\}$, which is close to the solution, but would fail to satisfy *condition*1.

To take all such cases into account, we can flatten the *givenAnswer* (which does not affect a fully correct answer) in *condition*4. To define one more class of partially correct answers, in *condition*5 we can compare the list elements in the *givenAnswer* to the solution and set a threshold, here at least six common elements, that indicates that the answer is similar to the solution. Obviously, it is up to the educator to decide whether or how many conditions to add for

partially correct answers, and how strict they should be.

After setting up the five conditions as shown in Figure 3.4.4, we can now define the scoring rules. When all the conditions are jointly $True$ the answer is fully correct and we can award full (here five) points in $scoringRule1$. Whenever $condition1$ yields $False$ we know that the answer is not fully correct. To qualify as a partially correct answer we could require that the number of characters is less than 19 and at least one of the last two conditions are $True$. In that case we could award three points ($scoringRule2$). Similarly, when the output is correct but the number of characters is too high, we could award three points ($scoringRule3$). When the answer is literally taken from the question we could still award one point (after all it satisfies both requirements) with $scoringRule4$. An answer that does not yield $True$ for any of the conditions is considered clearly wrong and gives no points ($scoringRule5$), the same as when no answer is submitted ($scoringRule6$). Results from the previous discussion are summarized in Table 3.4.4.

3.4.3 Interactive response

With the previously discussed answer types we can cover a wide range of different questions. However, all the examples so far were static— that is, the answer interface does not react to the user's action (besides displaying the input or selection). The last answer type allows educators to create interactive interfaces.

For this answer type there are two main fields of application: graphical answers and dynamic calculations. In many academic fields graphical representations of models are used to build intuition and as an abstraction from concrete examples. When it comes to assessment, typically students spend a lot of time drawing those graphical representations—only for modifying or adding one aspect of it to answer the question. Consider an example from economics where the student is asked how a certain external shock affects the equilibrium of demand and supply. Instead of asking the student to draw a coordinate system with the initial curves, we can provide the graphical interface to let them move the curves to answer the question.

In traditional exams of quantitative subjects, students are usually allowed to use a calculator. This allows them to work on more complex questions as they don't have to do all the calculations manually. The *Interactive Response* answer type allows educators to provide "custom calculators" to use when answering the questions.

This answer type is not widely used in education yet as developing interactive interfaces requires programming skills. While there is technical progress in simplifying programming, there are libraries such as the Wolfram Demonstrations Project that already offer thousands of freely available examples for educational purposes.

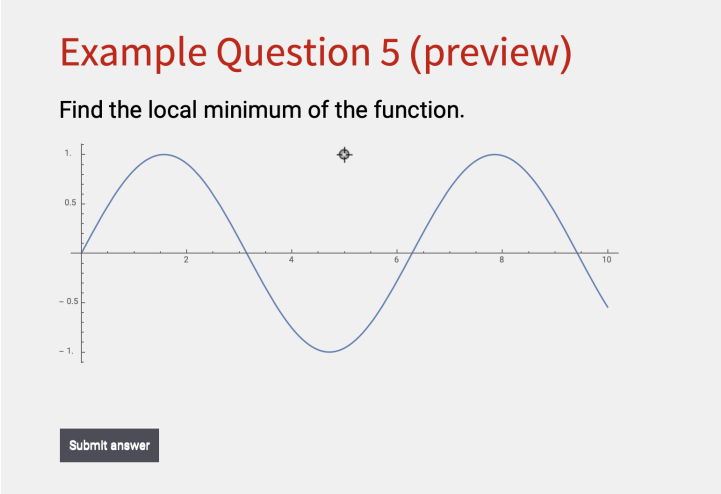
The core idea of the *Interactive Response* answer type is to take any interactive interface, such as Manipulate or DynamicModule, and (dynamically)

track parameters of interest to grade the answer based on conditions. In the next example question we will see how to set up and grade an interactive type answer.

Example Question 5

In the last example question we will discuss a graphical interface for the following task: "Find the local minimum of the function." Figure 3.4.5 gives a preview of the interactive user interface. In this case we use a locator to point to the local minimum of the sine function shown to the student.

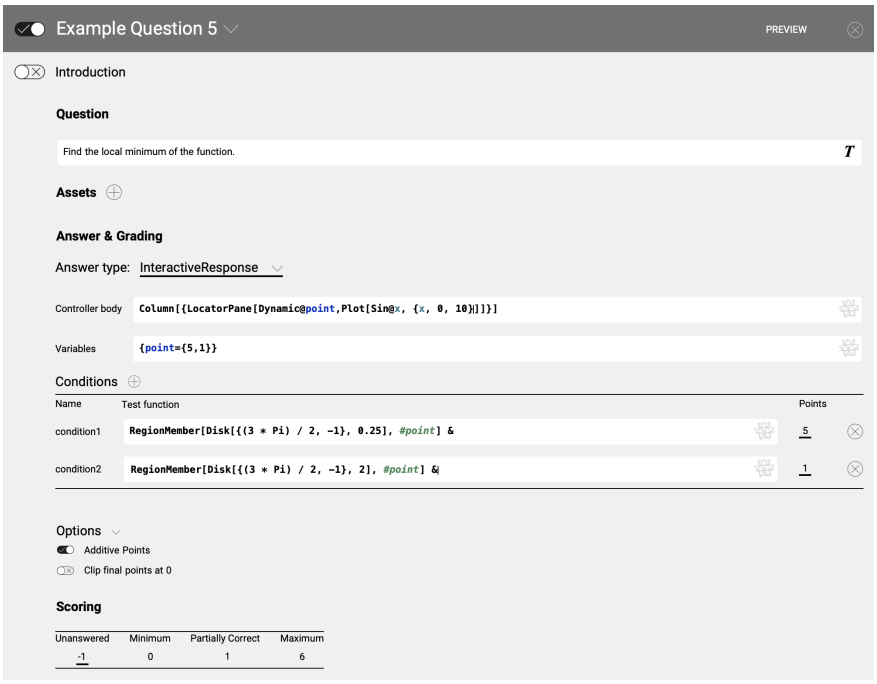
Figure 3.4.5: Example Question 5 preview



Since this answer type involves some coding of the interface, let's start with the setup as illustrated in Figure 3.4.6. The controller body contains the standalone interface—that is, the plot of the sine function and the dynamic locator. To grade the student's answer we need to track the position of the locator (at the time the answer is submitted). In our example the only dynamic element is *point*, but in more complex examples we would typically have several dynamic variables. We therefore need to state the variables that should be tracked. Here we can also set initial values for the dynamic variables.

To grade this question the conditions are now stated based on the variables (rather than *givenAnswer* like in the other examples). The solution to the question is the pair of point coordinates of the sine function at its minimum—so, $\{\frac{3}{2}\pi, -1\}$. To move the locator to this exact point is almost impossible (this is an open-ended question). We therefore need to define an error tolerance in the conditions. For our example we can check if the locator was moved into the region defined by a disk with radius 0.25 around the exact point (*condition*1). In this example it is difficult to think of a condition for a partially correct answer as the minimum is defined by a unique point (or region around it). One may argue

Figure 3.4.6: CREO setup of Example Question 5



that it's not necessary to award partial credits in this case. For demonstration purposes let's set up a second condition with a much larger error tolerance. Here the idea is that moving the locator closer to the minimum, relative to the initial point of the locator, could be interpreted as a vague understanding of the concept at least. Thus, setting the radius to 2.0 (the distance between the y-values of the initial point and the solution) in *condition*2 will allow us to detect if the locator was moved in the right direction.

With regard to the grading we can use additive consolidation, like in Example Question 3. Once the locator is moved toward the minimum (*condition*2) we award one point (*scoringRule*2). If then—in addition—the locator is moved sufficiently close to the exact minimum (*condition*1) we award five more points (*scoringRule*1). Results from the previous discussion are summarized in Table 3.4.5.

3.5 Grading

3.5.1 Grading philosophy and standards

Before we get to the grading of our example questions we need to discuss fundamental concepts of grading. A standardized, software-based assessment and evaluation system requires strict consistency with regard to the naming conventions it uses, and the interpretation of results it produces, if it aims to be

Table 3.4.5: Example Question 5: Interactive Response

Question	Find the local minimum of the function.	
Options	Additive points ⇾ True, Clip points at 0 ⇾ False	
Conditions		
condition1	*point* is within a radius of 0.5 around the minimum	
condition2	*point* is within a radius of 2.0 around the minimum	
Scoring Rules		Points
scoringRule1	condition1 is $True$	5
scoringRule2	condition2 is $True$	1
scoringRule3	none of the previous scoringRules is $True$	0
scoringRule4	no answer was given	-1

a general solution that many educators in different fields of higher education can apply. For this reason we will begin at the lowest level—that is, a single question or task—and derive four classes of possible outcomes. Based on this framework, we look at aggregated assessments to define the basic grading units to be used in SYLVA. To enable the handling of requests and (grading) errors, we will discuss how grading should be adjusted at the end of this section.

The four possible assessment outcome classes

When we are faced with the challenge of evaluating answers—that is to say, to which extent the quality of an answer is acceptable—we usually observe the following: while it is clear whether an answer is fully acceptable or clearly unacceptable, it is much harder to distinguish states in between, including whether an answer is *still acceptable* or *almost unacceptable*. Note that most grading schemes—numerical or letter-based—differentiate several performance levels. When such a grading scheme is applied on the question or task level, it suggests that the outcomes it generates are the product of an objective and precise evaluation process, and, therefore, possess the same attributes. In fact, however, with any additional distinction with regard to the grading it gets harder to come up with objective criteria that can be checked for. To illustrate the problem, consider the example of a classical multiple choice question with three alternative choices. Even when we distinguish all of the eight possible combinations of choices in the evaluation, we cannot map the evaluation result to a grade on a scale which consists of more than eight steps without further transformations or rules.

Therefore, Rapaport (2011) suggests—for philosophical and practical reasons—using a *triage* grading system; that is to say, a single answer should only either be:

- unanswered

- wrong
- partially correct, or
- correct

The main advantage of this approach is that three out of the four outcomes are easily distinguishable, and the fourth—*partially correct*—covers all other cases. While it is questionable whether *unanswered* and *wrong* should be distinguished from one another, in particular for a software-based assessment approach this is tracked automatically. SYLVA gives educators the flexibility to distinguish between these two outcomes by allowing them to set up so called scoring rules.

While *correct* and *wrong* answers can be clearly differentiated, there can still be several of each type. This is intuitive for *wrong* answers, but holds—at least for some questions—for *correct* answers too. Thus, instead of an attribute for only a single case (answer), we can think of outcome *classes*. This applies to *partially correct* answers, with one distinction: we can have different levels of *partially correct* answers with regard to the scoring. This can be seen as an extension of the triage grading approach, made possible by a systematic, condition-based grading approach. We will discuss this approach in detail in Section 3.5.8—for now, it is only important to note that this does not violate the concept of fixed outcome classes.

Grading units

Some educators use grades on the assessment level. For similar reasons as in the case of using grades on the question level, I will argue that this is not a good, and this for the following reasons: First, it is confusing to distinguish the same—by definition—grades on different levels. Second, this approach is prone to rounding errors, as grades are usually discrete. Third, grades have a signification, or—in other words—a meaning, often associated with ethical judgment. A *good* student is usually associated with a successful career path, is probably judged as being more trustworthy by many in society, and is maybe even more likely to get a loan from a bank. But then, what does it mean when this student was *excellent* in one assessment and *poor* in another one—in the same course? To avoid such interpretation issues, grades should be used only on higher assessment levels—that is, on the course level. Note that education institutions and systems share no common ground with regard to naming conventions. We will, therefore, define four evaluation outcomes:

- Score (question level)
  Number of points received for an answer to a question in an assessment.
- Performance (assessment level)
  Relative success—measured in percent—in an assessment with regard to the ratio of achieved score to maximum score.

- Grade (course level)
  Classification of learning success in a course according to particular standardized definitions. A grade typically has a grade value and a signification.

- Certificate (study program level)
  Degree that is obtained after successfully completing a study program.

Note that the level of abstraction decreases with the assessment level—that is to say, while a degree is a concrete accomplishment or signal that one has completed a particular study program successfully, the interpretation of a grade is harder without further context regarding the course. One's performance in an assessment can only be interpreted within the context of a course and scores might even vary between assessments in the same course.

Adjusting grading

Grading on a curve—that is, transforming evaluation results to fit a pre-defined distribution—is one of the most popular approaches used to adjust grading in cases in which the initial evaluation does not lead to the desired results (Brookhart et al., 2016). The main advantages of this approach are: First, it leads to normalized results—a desirable outcome especially when there are several instructors in large courses. Second, it is easy to apply for educators because the transformation is only based on aggregated performance measures. On the downside, grading on the curve has several major flaws: First, it is likely to violate fairness norms because—based on the peer group a student is evaluated against—it is no longer guaranteed that the same performance (answers) leads to the same evaluation: why should a *good* student in a class of otherwise *excellent* students get a worse grade than an *mediocre* student in a class of otherwise *poorly performing* students (Clio, 2003)? Second, it introduces competitiveness among students (Krumboltz and Yeh, 1996), as opposed to collaborative learning—the latter having become one of the most promising learning approaches (Slavin, 2014, 1980; Kirschner et al., 2009). Third, grading on a curve distorts the interpretability of performance measures, since they are no longer related to learning objectives but only to the performance of the peer group (Rapaport, 2011). When it is no longer clear that a particular performance—that is, answering a portion of an assessment correctly—leads to a certain evaluation result, incentives for learning may fade as more randomness is introduced. Fourth, it incentivizes educators to design assessments and instructions poorly (Krumboltz and Yeh, 1996). For grading on a curve it is sufficient to get *any* level of distinction in the performance distribution—since any distribution can be fitted to the curve. Thus, educators (in minimizing their effort) only need to make sure that an assessment is hard or ambiguous enough so that at least some students will succeed or fail, in at least one part of it—even by means of randomness.

From the previous discussion we can derive the following implications: An assessment system—by means of its design—can affect fairness by allowing or hampering certain behaviors of its users. In concrete terms, educators can be incentivized to put more effort into assessment preparation by making it less convenient to adjust grades with a single transformation or click. Note that any transformation of grading can potentially jeopardize fairness or the interpretability of the evaluation. Therefore, as a principle, grading should be adjusted on the lowest possible assessment level—that is, in the least intrusive way. For the time being only conditions, scores, scoring rules, and assessment weights (which will be explained in more detail in the following section) can be adjusted in SYLVA. As we will see in Section 4.4, we equip educators with flexible tools with which to precisely target specific problems of evaluation—without violating fairness norms—rather than providing a one-size-fits-all (curves) approach.

3.5.2 The automated grading process

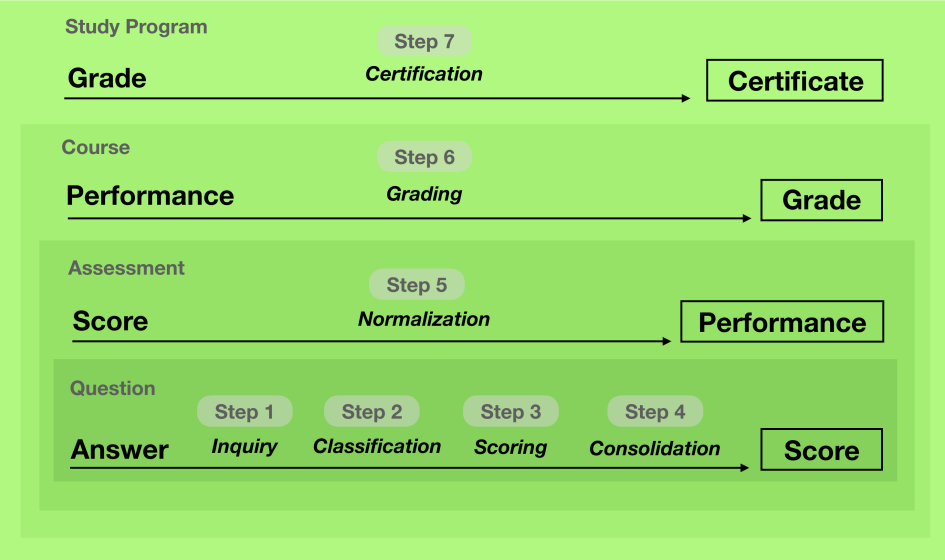
When we refer to *grading* this typically includes the entire process starting with the review of answers from students through the assignment of points to the determination of final grades. What is condensed in a single term is actually a complex process. While educators develop individual approaches to handling grading as they gain experience, for an automated grading solution the process needs to be standardized if it is to attain consistent and robust results.

One of the biggest challenges in educational assessment is the flexibility to handle modifications, such as changing points, weights, or accepted answers, either caused by human error or in the form of individual exceptional cases. In contrast to human graders, a software system cannot offer case-by-case solutions or any "pragmatic" approaches—all such use cases need to be predefined. Therefore it is vital to cover the entire process, including all steps from processing a single answer to awarding certificates in a study program.

When large courses with hundreds of students are graded, the work is often split among several graders, which typically cover either only a few questions or one of the steps in the grading process. This needs a coordinated process if consistent results are to be delivered. Similarly, in SYLVA the grading consists of seven steps, which are shown in Figure 3.5.1. Intuitively, at each assessment level it needs at least one process—so, for example, to get from the *performances* of assessments to a final *grade* in a course.

At the lowest level—from *Answer* to *Score*— the process consists of four steps. We will discuss them in detail in the following sections, but at this stage one may already ask why several steps are needed. The reason is to be able to handle the aforementioned modifications. Conceptually, the process needs to be split, such that the options and parameters set in the creation of the assessment can be adjusted independently, to cover all possible use cases and exceptions. This is also useful from a computational perspective when we want to calculate the grading results as fast as possible, and compute efficiently. If the grading

Figure 3.5.1: Grading steps overview



process was designed as a single procedure, changing an assessment weight, for instance, would require us to re-compute all previous steps again. With intermediate grading results, modifications will only affect subsequent steps.

Since automated grading requires this high level of formalization, we will define and use the terms and notation presented in Table 3.5.1 for the next chapters. Note that most of the terms have been invented especially for the grading approach presented here as there are—to the best of my knowledge—no established naming conventions, and in human grading these processes are not distinguished.

Assessment simulation

In the previous sections we derived the setup for five different example questions. In this section we will now simulate the grading to discuss the seven steps in the process of grading shown in Figure 3.5.1. To simulate the entire flow, consider the following setup:

- There is one study program $\mathcal{P}$ where five students $\mathcal{S}_1$, ..., $\mathcal{S}_5$ can attain a certificate $Cert$ iff the average grade in the courses $\mathcal{C}_1$ and $\mathcal{C}_2$ is at least 4.0.

- Course $\mathcal{C}_1$ has a passing threshold of 60% with two assessments $\mathcal{A}_1$ and $\mathcal{A}_2$. Course $\mathcal{C}_2$ has a passing threshold of 60% with two assessments $\mathcal{A}_1$ and $\mathcal{A}_3$.

Table 3.5.1: Grading terms overview

Notation	Name	Definition
$\mathcal{Q}$	question	question or task of an assessment
$\mathcal{A}$	assessment	assessment of a course
$\mathcal{C}$	course	course of a study program
$\mathcal{P}$	study program	sequence of courses
$\mathcal{S}$	student	studying person
a	answer	given answer to a question
c	condition	Section 3.3.3
cM	consolidation method	Definition 3.5.6
cR	consolidation rule	Section 3.5.6
sR	scoring rule	Definition 3.5.2
sS	scoring scheme	Definition 3.5.2
nR	normalization rule	Section 3.5.7
w	assessment weight	Sections 3.2.2 & 3.5
gS	grading scheme	Definition 3.5.9
gR	grading rule	Section 3.5.8
cS	certification scheme	Section 3.5.9
IR	inquiry result	Definition 3.5.1
CR	classification result	Definition 3.5.3
SR	scoring result	Definition 3.5.4
$Scor$	score (consolidation result)	Definition 3.5.5
$Perf$	performance (normalization result)	Definition 3.5.7
$Grad$	grade (grading result)	Definition 3.5.8
$Cert$	certificate (certification result)	Definition 3.5.10

- Assessment $\mathcal{A}_1$ has a weight $w_{\mathcal{A}_1}$ of 20% and consists of questions $\mathcal{Q}_1$ and $\mathcal{Q}_4$. Assessment $\mathcal{A}_2$ has a weight $w_{\mathcal{A}_2}$ of 80% and consists of questions $\mathcal{Q}_1$, $\mathcal{Q}_3$, and $\mathcal{Q}_4$. Assessment $\mathcal{A}_3$ has a weight $w_{\mathcal{A}_3}$ of 80% and consists of questions $\mathcal{Q}_1$, $\mathcal{Q}_2$, $\mathcal{Q}_3$, $\mathcal{Q}_4$, and $\mathcal{Q}_5$.
- Questions $\mathcal{Q}_1$ to $\mathcal{Q}_5$ are taken from Section 3.4 in their respective order.

An overview of students' answers is provided in Table 3.5.2. The simulation is constructed to cover different cases of answers. In the next section we will grade the simulation in seven steps.

3.5.3 Step 1: Inquiry

The automated grading process starts at the question level with the *inquiry*:

Definition 3.5.1 (Inquiry)**.** *Inquiry is the process of assigning truth values (Boolean values) separately to each of $i \geq 1, i \in \mathbb{N}$ conditions c_i of a question $\mathcal{Q}$ (of an assessment $\mathcal{A}$) with regard to a particular answer a. A condition c_i*

Table 3.5.2: Simulated answers

Answers	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	True	{1,2,3}	Paris	Range[9]	{(3 * Pi) / 2, -1}
$\mathcal{S}_2$	True	{1,2}	Amsterdam	Table[{i}, {i, 9}]	{5,-1}
$\mathcal{S}_3$	True	{1,2,4}	Tokyo	NestWhileList[# + 1 &, 1, # <9 &]	{0,1}
$\mathcal{S}_4$	False	{1,4,5}	London	{1,2,3,4,5,6,7,8,9}	{5,0}
$\mathcal{S}_5$	no answer	{4,5}	Glasgow	NestList[# + 1 &, 4, 8]	no answer

either yields True, i.e., when the evaluation of the inquiry function $c_i(a)$ gives True, or False in all other cases. The inquiry result $IR_{\mathcal{A},\mathcal{Q}}(c,a)$ is a list of i truth values.

Note that each $c_i(a)$ can be represented as $f(lhs, rhs)$ where lhs is an expression containing the answer and rhs an expression that contains the condition. Then there are several possible functions f to assign a Boolean value to $c_i(a)$. Checking if both sides are the same (exact correspondence) using SameQ (lhs===rhs) can be too restrictive when there are several expressions that can be interpreted as equal (but not the same) in the context of the question. Checking if both sides are equal using Equal (lhs==rhs) can fail to yield Boolean values when the function remains symbolic, if any side yields *Indeterminate*, or when the evaluation exceeds the evaluation time or memory limits of the system. To avoid these problems f can be chosen to assume that the result is *False* whenever it is not clear or explicitly *True*—that is, `TrueQ[lhs==rhs]`. This is sufficient to make the inquiry function robust against the aforementioned cases. For a platform that allows both educators and students to ask and answer open-ended questions, robustness is crucial. However, this comes at the price of a few sacrifices:

1. When evaluating $c_i(a)$ exceeds the system time limit, the condition will always yield *False*, even when the answer is correct. To illustrate this problem, consider the following example:
`TrueQ[TimeConstrained[Pause[2], 1] == Pause[2]]`.
This will yield *False* as long as the time constraint is too low, and *True* once it is sufficient, in this case for example `TimeConstrained[Pause[2], 3]`.

2. To check for indeterminate solutions equality is not sufficient. As soon as either side is indeterminate, evaluating $c_i(a)$ will always yield *False*. Consider the example `TrueQ[0/0 == 0/0]`, which yields *False*, but could be verified as *True* only by using SameQ (0/0 === 0/0)—that is, exact correspondence.

While the second problem is a rare case with little practical relevance, the first problem could lead to faulty inquiry results. To prevent this, the system can be configured to support reasonable evaluation times. However, due to physical, technical, and budgetary limitations, there will always be cases that can lead to faulty inquiry results. This can be solved by checking all evaluations and returning—that is, not accepting—those that take too long to the user, asking them to change their answer (student), or the question (educator).

A special case arises for the inquiry of unanswered questions. From a computational perspective it does not make sense to spend resources on checking conditions for unanswered questions since the inquiry function would be called without an *lhs* argument. Conceptually it is also not possible to check conditions for unanswered questions. Consider the case of a task like "Leave this question unanswered". To answer the question the student would need to submit *no answer*, which is a self-contradiction. Since some answer types such as *Interactive Response* require a default state (values) the answer submission procedure is necessary to distinguish between an answer that equals the default and no answer. The inquiry is not, therefore, performed for unanswered questions.

Table 3.5.3: Inquiry results for Example Questions 1 & 2

a	$\mathcal{Q}_1$	$\mathcal{Q}_2$
$\mathcal{S}_1$	choice1	choices 1,2,3
$\mathcal{S}_2$	choice1	choices 1,2
$\mathcal{S}_3$	choice1	choices 1,2,4
$\mathcal{S}_4$	choice2	choices 1,4,5
$\mathcal{S}_5$	no answer	choices 4,5
IR	$\mathcal{Q}_1$	$\mathcal{Q}_2$
$\mathcal{S}_1$	{True, False}	{True, True, True, True, True}
$\mathcal{S}_2$	{True, False}	{True, True, False, True, True}
$\mathcal{S}_3$	{True, False}	{True, True, False, False, True}
$\mathcal{S}_4$	{False, True}	{True, False, False, False, False}
$\mathcal{S}_5$	—	{False, False, False, False, False}

Now we can look at the inquiry results for the selection questions in Table 3.5.3. For $\mathcal{Q}_1$ we have two conditions and consequently a list of two truth values for each given answer. By the nature of true/false (select one) questions, as soon as one condition is $True$ the other one (all others) must be $False$ since it is a binary decision (single choice). In $\mathcal{Q}_2$ we have a *Selection* question with different combinations of choices represented by the simulated answers. For $\mathcal{S}_1$ three choices lead to $True$ for all conditions since the last two conditions are $True$ when the corresponding choices are *not* selected. Accordingly, selecting only those last two choices will yield $False$ for all conditions for $\mathcal{S}_5$.

Let's look at the inquiry of the remaining three questions now. Table 3.5.4 provides an overview of the results. The inquiry of $\mathcal{Q}_3$ involves three conditions:

1. Except Glasgow and Tokyo all other cities are within 300 miles of London and therefore yield $True$ for the first condition.

2. Paris, Amsterdam, and Tokyo are not within the UK and therefore give $True$ for the second condition.

3. The last condition combines the first condition and additionally rules out

London (to avoid awarding too many partial points for repeating the city from the question text). Only Paris and Amsterdam satisfy the third condition while the other three answers yield $False$.

Table 3.5.4: Inquiry results for Example Questions 3, 4 & 5

a	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	Paris	Range[9]	{(3 * Pi) / 2, -1}
$\mathcal{S}_2$	Amsterdam	Table[{i}, {i, 9}]	{5,-1}
$\mathcal{S}_3$	Tokyo	NestWhileList[# + 1 &, 1, # <9 &]	{0,1}
$\mathcal{S}_4$	London	{1,2,3,4,5,6,7,8,9}	{5,0}
$\mathcal{S}_5$	Glasgow	NestList[# + 1 &, 4, 8]	no answer
IR	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	{True, True, True}	{True, True, True, True, True}	{True, True}
$\mathcal{S}_2$	{True, True, True}	{False, True, True, True, True}	{True, True}
$\mathcal{S}_3$	{False, True, False}	{True, True, False, True, True}	{False, False}
$\mathcal{S}_4$	{True, False, False}	{True, False, True, True, True}	{False, True}
$\mathcal{S}_5$	{False, False, False}	{False, True, True, False, True}	—

To check the conditions for the coding question $\mathcal{Q}_4$ all answers get evaluated first. In Table 3.5.5 we can see the intermediate output of the evaluation and a count of the code characters. The inquiry of the questions consists of five conditions:

1. All but the answers from $\mathcal{S}_2$ and $\mathcal{S}_5$ evaluate to the required output.

2. Only the answers of $\mathcal{S}_4$ are literally the same as the list in the question statement and therefore leads to $False$.

3. The number of characters is less than 20 for all but the answer from $\mathcal{S}_3$.

4. Applying Flatten on the answers yields the required output for all answers but the one from $\mathcal{S}_5$. Note that Flatten only removes brackets in nested lists and therefore the answers that satisfy the first condition will not be affected.

5. All the evaluated (and flattened) answers have at least 5 elements in common with the required list.

In this example we can see how various characteristics can be computed based on both the initial answer and interpreted (evaluated) versions. This concept—similar to feature extraction—allows us to acquire data about the answers systematically to distinguish different states with regard to the scoring and grading performed in the next steps.

The last question $\mathcal{Q}_5$ comes with two conditions:

Table 3.5.5: Evaluated answers to Example Question 4

	Output(a)	Characters(a)
$\mathcal{S}_1$	{1,2,3,4,5,6,7,8,9}	8
$\mathcal{S}_2$	{{1},{2},{3},{4},{5},{6},{7},{8},{9}}	16
$\mathcal{S}_3$	{1,2,3,4,5,6,7,8,9}	26
$\mathcal{S}_4$	{1,2,3,4,5,6,7,8,9}	19
$\mathcal{S}_5$	{4,5,6,8,9,10,11,12}	18

1. The first condition checks if the locator was moved reasonably close to the exact point of the local minimum (approximately {4.7, -1}). Conceptually the error tolerance can be thought of as a circle of radius 0.5 around the point and any point within the area would be accepted. Thus, only the answers from $\mathcal{S}_1$ and $\mathcal{S}_2$ yield $True$ for the first condition.

2. The second condition defines a wider circle of radius 2.0. The answer from $\mathcal{S}_4$ is within the area, area, as are the previously mentioned answers. The answer from $\mathcal{S}_3$, however, does not satisfy any of the conditions. Since $\mathcal{S}_5$ has not provided an answer, no inquiry is performed.

To summarize this step, we get consistent lists of truth values for all provided answers and students. Note that the inquiry results do not allow any conclusions about the correctness of an answer at this stage. For an answer to be marked as correct, neither the number nor the order of conditions is relevant. As we have seen in several examples, conditions can be reversed with regard to their truth value in the initial setup.

Thus, not even the actual truth values of the inquiry result indicate correctness. The inquiry process consumes the most computational resources but the results are only intermediates for the actual grading. In the next step the inquiry results are classified for scoring. For that we need scoring rules.

Definition 3.5.2 (Scoring Rules and Scoring Schemes)**.** *A scoring rule sR_j is a statement composed of conditions c_i and logical operators that can evaluate to either $True$ or $False$. Each sR is associated with a single points value. The list of the j values is called the scoring scheme sS.*

3.5.4 Step 2: Classification

In Section 3.5.1 we discussed the four possible classes of outcomes. Depending on the type of question, we need to distinguish at least three classes—that is, *correct*, *wrong* and *unanswered*—to evaluate an answer. We can have arbitrarily many additional scoring rules for evaluating *partially correct* answers. The number of scoring rules is independent of the number of conditions.

Definition 3.5.3 (Classification)**.** *Classification is the process of assigning truth values to $j \geq 3, j \in \mathbb{N}$ scoring rules sR_j based on the inquiry result $IR_{\mathcal{A},\mathcal{Q}}(c,a)$. The classification result $CR_{\mathcal{A},\mathcal{Q}}(IR, sR)$ is a list of j truth values.*

Computationally the classification step is straightforward since scoring rules only include conditions and logical operators. The special case of unanswered questions is incorporated into the process with a dedicated scoring rule. The classification is therefore robust and will always yield consistent lists of truth values for *any* type of answer.

Note that the classification only takes the inquiry results IR and scoring rules sR as arguments. It is already computationally independent of the conditions and answers. This principle is used for all steps of the grading process to allow modifications to be applied efficiently. To illustrate this, consider the case of an ex post adjustment of points awarded for a particular question—that is, a modification of a scoring rule. Instead of running the resource-intense inquiry again, based on the inquiry results only the classification and following steps have to be recalculated. This comes at the price of more data being created and associated read and write times but usually the performance goal for grading is speed and the amount of data for the intermediates is limited to a few bytes per student.

In Table 3.5.6 we see the classification results for the first two example questions. For a more compact view, we switch to common abbreviations for our truth values, that is T for *True* and F for *False*.

Table 3.5.6: Classification results

IR	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	{T, F}	{T, T, T, T, T}	{T, T, T}	{T, T, T, T, T}	{T, T}
$\mathcal{S}_2$	{T, F}	{T, T, F, T, T}	{T, T, T}	{F, T, T, T, T}	{T, T}
$\mathcal{S}_3$	{T, F}	{T, T, F, F, T}	{F, T, F}	{T, T, F, T, T}	{F, F}
$\mathcal{S}_4$	{F, T}	{T, F, F, F, F}	{T, F, F}	{T, F, T, T, T}	{F, T}
$\mathcal{S}_5$	—	{F, F, F, F, F}	{F, F, F}	{F, T, T, F, T}	—
sR	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
sR_1	c_1	$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge c_5$	c_1	$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge c_5$	c_1
sR_2	c_2	$c_1 \vee c_2 \vee c_3 \vee c_4 \vee c_5$	c_2	$c_3 \wedge (c_4 \vee c_5)$	c_2
sR_3	unanswered	$\neg(c_1 \vee c_2 \vee c_3 \vee c_4 \vee c_5)$	c_3	$c_1 \wedge c_2 \wedge \neg c_3$	$\neg(c_1 \vee c_2)$
sR_4		unanswered	$\neg(c_1 \vee c_2 \vee c_3)$	$\neg c_2$	unanswered
sR_5			unanswered	$\neg(c_1 \vee c_2 \vee c_3 \vee c_4 \vee c_5)$	
sR_6				unanswered	
CR	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	{T, F, F}	{T, T, F, F}	{T, T, T, F, F}	{T, T, F, T, F, F}	{T, T, F, F}
$\mathcal{S}_2$	{T, F, F}	{F, T, F, F}	{T, T, T, F, F}	{F, T, F, F, F, F}	{T, T, F, F}
$\mathcal{S}_3$	{T, F, F}	{F, T, F, F}	{F, T, F, F, F}	{F, T, T, F, F, F}	{F, F, T, F}
$\mathcal{S}_4$	{F, T, F}	{F, T, F, F}	{T, F, T, F, F}	{F, F, F, T, F, F}	{F, T, F, F}
$\mathcal{S}_5$	{F, F, T}	{F, F, T, F}	{F, F, F, T, F}	{F, T, F, F, F, F}	{F, F, F, T}

For question $\mathcal{Q}_1$ we have only two elements in the inquiry result but three scoring rules. Therefore we get lists of three truth values from the classification

for this question. Comparing the five questions we can see that the number of scoring rules is independent of the number of inquiry results.

To generate the classification result for student $\mathcal{S}_1$ we can now evaluate the scoring rules sequentially. Since the first condition was $True$ the first scoring rule yields $True$ as well, and vice versa for the second scoring rule. The third scoring rule checks if the question was *unanswered*, which would only be the case if there is no inquiry result. For student $\mathcal{S}_1$ this would yield $False$ for the last classification result element.

For all other questions we have cases with more complex scoring rules. For example, sR_1 of $\mathcal{Q}_2$, which checks if all five conditions are jointly $True$. While scoring rules can involve all logical operators, the computations are always sequential evaluations of the scoring rules based on the inquiry results.

With the classification we linked the neutral inquiry results to evaluative scoring rules. This is the basis for assigning points in the next step. However, based on the classification results we still cannot interpret if a particular question was answered correctly. The reason for this is the flexibility of the scoring rules. If there were always only exactly four scoring rules (for the four possible outcomes), we could immediately assign a state to each answer. Since we can have arbitrarily many scoring rules (for partially correct answers) and the order of the scoring rules is not fixed, it is not possible to tell which scoring rule(s) will lead to a correct answer.

3.5.5 Step 3: Scoring

In the next step we will now use the classification results to assign points to the students' answers.

Definition 3.5.4 (Scoring)**.** *Scoring is the process of assigning points according to the scoring scheme sS to all of the $j \geq k \geq 1, k \in \mathbb{N}$ elements of the classification result $CR_{\mathcal{A},\mathcal{Q}}(IR, sR)$ that are $True$. The scoring result $SR_{\mathcal{A},\mathcal{Q}}(CR, sS)$ is a list of k numbers.*

Computationally, the scoring step is simply a replacement of truth values by numbers. The scoring results are given in Table 3.5.7. For a better overview the list elements for $False$ values of the classification results are preserved and marked by dashes. For $\mathcal{Q}_2$ student $\mathcal{Q}_1$ has $True$ for the first two and $False$ for the last two elements in the classification result. This leads to a scoring result of {3,1} because those are the first two values of the scoring scheme.

Although we have point values for all answers after completing this stage, the interpretation of the results with regard to the four possible states of correctness is still not unambiguous for all questions. For the first question we see three students with 2 points and one student with -2 and 0 points, respectively. From the scoring scheme it is clear that the three students with 2 points got it *fully correct*, and thus were awarded maximum points for this questions. Student

$\mathcal{S}_5$ did not answer the question and student $\mathcal{S}_4$ gave a wrong answer and got minimum points. For this question there are no partial points.

For all the other questions some students will get partial points. As shown in Table 3.5.7 there is always at least one student with several numbers in the scoring results. This will occur whenever several scoring rules yield $True$. Thus, the scores have to be consolidated before we get conclusive results.

Table 3.5.7: Scoring results

CR	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	{T, F, F}	{T, T, F, F}	{T, T, T, F, F}	{T, T, F, T, F, F}	{T, T, F, F}
$\mathcal{S}_2$	{T, F, F}	{F, T, F, F}	{T, T, T, F, F}	{F, T, F, F, F, F}	{T, T, F, F}
$\mathcal{S}_3$	{T, F, F}	{F, T, F, F}	{F, T, F, F, F}	{F, T, T, F, F, F}	{F, F, T, F}
$\mathcal{S}_4$	{F, T, F}	{F, T, F, F}	{T, F, F, F, F}	{F, F, F, T, F, F}	{F, T, F, F}
$\mathcal{S}_5$	{F, F, T}	{F, F, T, F}	{F, F, F, T, F}	{F, T, F, F, F, F}	{F, F, F, T}
sS	{2, -2, 0}	{3, 1, -1, 0}	{1, 2, 1, 0, 0}	{5, 3, 3, 1, 0, 0}	{5, 1, 0, -1}
SR	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	{2, —, —}	{3, 1, —, —}	{1, 2, 1, —, —}	{5, 3, —, 1, —, —}	{5, 1, —, —}
$\mathcal{S}_2$	{2, —, —}	{—, 1, —, —}	{1, 2, 1, —, —}	{—, 3, —, —, —, —}	{5, 1, —, —}
$\mathcal{S}_3$	{2, —, —}	{—, 1, —, —}	{—, 2, —, —, —}	{—, 3, 3, —, —, —}	{—, —, 0, —}
$\mathcal{S}_4$	{—, -2, —}	{—, 1, —, —}	{1, —, —, —, —}	{—, —, —, 1, —, —}	{—, 1, —, —}
$\mathcal{S}_5$	{—, —, 0}	{—, —, -1, —}	{—, —, —, 0, —}	{—, 3, —, —, —, —}	{—, —, —, -1}

3.5.6 Step 4: Consolidation

In this step we will finalize the grading on the question level. The results from the consolidation are provided in Table 3.5.8.

Definition 3.5.5 (Consolidation)**.** *Consolidation is the process of unifying scoring results $SR_{\mathcal{A},\mathcal{Q}}$ according to the consolidation method cM and assigning an assessment outcome based on the scoring scheme sS. Then, additional consolidation rules cR are applied. For each answer $a_{\mathcal{A},\mathcal{Q}}$ a pair consisting of a points value and the assessment outcome is obtained. The consolidation result $Scor_{\mathcal{A},\mathcal{Q}}(SR, sS, cM, cR)$ is called score.*

According to the definition above the consolidation process consists of three steps:

1. Consolidation of points to a single score.

2. Consolidation of assessment outcomes to a single statement.

3. Consolidation of additional rules.

In order to unify the points we need to consider the consolidation method cM for each question. Note that cM is set in the creation of the questions (Section 3.4) via the option *Additive points*.

Definition 3.5.6 (Consolidation Method)**.** *The consolidation method cM determines how the scoring results are unified. There are two consolidation methods: additive and holistic. For each student additive consolidation sums up all values from the scoring results SR. With holistic consolidation the maximum value of SR is picked.*

For question $\mathcal{Q}_1$ we have holistic consolidation and therefore we extract from each scoring result the maximum points. This gives the first element of the consolidation result. In $\mathcal{Q}_3$ we have additive consolidation. Here, the answers of students $\mathcal{S}_1$ and $\mathcal{S}_2$ have points associated with three scoring rules. After summing up all the points for each of the students we get 4 (=1 + 2 + 1), respectively. In contrast, for $\mathcal{Q}_4$ (holistic consolidation) the answer of student $\mathcal{S}_1$ satisfies several scoring rules too, but the one that yields the maximum of points is picked.

Table 3.5.8: Consolidation results

SR	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	{2, —, —}	{3, 1, —, —}	{1, 2, 1, —, —}	{5, 3, —, 1, —, —}	{5, 1, —, —}
$\mathcal{S}_2$	{2, —, —}	{—, 1, —, —}	{1, 2, 1, —, —}	{—, 3, —, —, —, —}	{5, 1, —, —}
$\mathcal{S}_3$	{2, —, —}	{—, 1, —, —}	{—, 2, —, —, —}	{—, 3, 3, —, —, —}	{—, —, 0, —}
$\mathcal{S}_4$	{—, -2, —}	{—, 1, —, —}	{1, —, —, —, —}	{—, —, —, 1, —, —}	{—, 1, —, —}
$\mathcal{S}_5$	{—, —, 0}	{—, —, -1, —}	{—, —, —, 0, —}	{—, 3, —, —, —, —}	{—, —, —, -1}
cM	holistic	holistic	additive	holistic	additive
Scor	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	{2, correct}	{3, correct}	{4, correct}	{5, correct}	{6, correct}
$\mathcal{S}_2$	{2, correct}	{1, partially correct}	{4, correct}	{3, partially correct}	{6, correct}
$\mathcal{S}_3$	{2, correct}	{1, partially correct}	{2, partially correct}	{3, partially correct}	{0, wrong}
$\mathcal{S}_4$	{-2, wrong}	{1, partially correct}	{1, partially correct}	{1, partially correct}	{1, partially correct}
$\mathcal{S}_5$	{0, unanswered}	{-1, wrong}	{0, wrong}	{3, partially correct}	{-1, unanswered}

With additive consolidation the scoring rules are split such that several components contribute to the overall correctness of an answer. In example question $\mathcal{Q}_3$ these are the scoring rules associated with the conditions for the distance and the country property. The points associated with the scoring rules can then be interpreted as the weights or importance of each of the components with regard to the solution of the question. In this example it is more important to name a city that is outside the UK than to name a city within a distance of 300 miles. Therefore, the answer of $\mathcal{S}_3$, Tokyo (which does not satisfy the distance condition), would only give partial points, but still more points than the answer of $\mathcal{S}_4$, London (which is not outside the UK but is within the demanded distance).

For questions with holistic consolidation the intuition is usually that the scoring rules represent different quality levels of an answer. A code that yields the demanded list in $\mathcal{Q}_4$ has a higher quality than a code that needs additional transformations such as Flatten. Recall that the motivation for setting up those additional conditions and scoring rules was to catch answers that are not fully correct, but still get close to the solution. In this example, some of the scoring

rules imply others—that is, if sR_1 is $True$, then sR_2 is $True$ as well, but not vice versa. On the one hand, this explains why we can get several scoring rules matching for a single answer. On the other hand, this explains why we always pick the maximum of the scoring results in holistic consolidation, or, the highest quality level a particular answer represents.

To consolidate the assessment outcome, each answer must be associated with exactly one of four possible outcome classes: *unanswered*, *correct*, *wrong*, or *partially correct*. To compute the assessment outcome the following checks are conducted sequentially and the case that matches first is the final result. Obviously, the answer is classified as *unanswered* if no answer has been actively submitted, which is indicated when the last scoring rule is $True$. Otherwise, the answer is classified as *correct* if the scoring rules that are associated with the maximum of the points (holistic consolidation) or the sum of the values of sS minus the value for unanswered questions (additive consolidation) are $True$. An answer which is associated with the minimum of the values of sS for answered questions is classified as *wrong*. All other answers are *partially correct*.

Note that this procedure is required since neither the scoring rules nor the conditions are associated with assessment outcomes in the initial question setup. As there can be several scoring rules associated with the same values (sS), all assessment outcome classes except *unanswered* can cover several different answers. However, *correct* and *wrong* answers will always yield the same points. Only *partially correct* answers can vary in terms of points.

The consolidation is distinct and robust for all questions that are graded with holistic consolidation. For additive consolidation the consolidation of points is distinct, but the assessment outcomes can only be derived reliably when the scoring rules are non-contradictory. To understand the problem consider the following abstract example: *Find an instance of X which satisfies A and B.* Assume the question was set up to be graded using $c_1 = A, c_2 = B, sR_1 = c_1, sR_2 = c_2$. For the case that A implies *not* B then no X exists that satisfies both conditions at the same time. With additive consolidation only one of the conditions could possibly match and the maximum points would be miscalculated. This case is not critical because the example represents a question that has no solution. However, with a slight extension of the example we run into the same problem. Now consider the example: *Find an instance of X that satisfies A and either B or C.* Assume the question was set up to be graded using $c_1 = A, c_2 = B, c_3 = C, sR_1 = c_1, sR_2 = c_2, sR_3 = c_3$. If B implies *not* C, the question can have a solution, but the consolidation would lead to false results with the additive method. The problem can be avoided by using either holistic consolidation (with different scoring rules) or by changing the scoring rules for the additive consolidation to $sR_1 = c_1, sR_2 = c_2 \vee c_3$, for instance.

The previous discussion demonstrates that the consolidation is always distinct (correct with regard to the score), but not robust with regard to human error in setting up the questions consistently. To protect educators from making

such mistakes it would be necessary to check all conditions against each other for contradictoriness. This is not implemented in the current version of SYLVA for the following reason: for any open-ended question the required logical checks might not be computationally feasible. Recall the example question $\mathcal{Q}_3$. To check whether the distance condition contradicts the country property condition, at least one instance needs to be found that satisfies both conditions. That means in the worst case all instances of the feasible set (here cities within 300 miles distance to London) would need to be checked. In other words, the computational effort required to verify a solution according to conditions and scoring rules is much lower than the effort required to find a solution. With independent evaluations of the conditions and scoring rules, the computational complexity grows linearly as opposed to exponentially in the case of cross-verification of conditions and scoring rules. For that reason solutions are only computed automatically for distinct questions with answer type Selection.

In the last step the consolidation rules cR are applied. Conceptually, these rules cover special cases and exceptions which are applied after the regular grading process has passed all previous steps. Recall the setup of the first example question, from Figure 3.4.1. The last option *Clip final points at 0* is an example of a consolidation rule. If the option was activated, the score for the answer from $\mathcal{S}_4$ would change from {-2, wrong} to {0, wrong}, effectively overwriting the corresponding points value in the scoring scheme. This option is only relevant for a few use cases (in this example it would not make much sense since it's a binary decision) and is therefore not discussed in greater detail. At this point it is important to pay attention to the sequence of the execution of the consolidation rules. In the example of $\mathcal{Q}_1$ the scoring rule for unanswered questions is associated with 0 points too. To prevent interference with the previous steps, the consolidation rules have to be applied at the end of the consolidation process. Further examples of consolidation rules include practical use cases such as exception handling for single students. In our simplistic simulation no such rules take effect.

Now, the question level grading is complete and we will move on to the assessment, course, and study program level in the next steps.

3.5.7 Step 5: Normalization

Once the grading of all single questions is finished, the results are combined to calculate the performance of the assessment.

Definition 3.5.7 (Normalization)**.** *Normalization is the process of combining scores $Scor_{\mathcal{A},\mathcal{Q}}$ of all $q \geq 1 \in \mathbb{N}$ questions to calculate a relative measure of the overall performance. Then, additional normalization rules nR are applied. The normalization result is a pair, consisting of a performance value and a corresponding assessment result, called performance $Perf_{\mathcal{A}}(Scor, nR)$.*

The normalization results are shown in Table 3.5.9. For a better overview

the composition of the assessments is provided in the middle part of the table. To compute the performance we need the maximum points for each question. For $\mathcal{A}_1$, questions $\mathcal{Q}_1$ and $\mathcal{Q}_4$ add up to 7 points and the performance of each student is calculated as points scored divided by the sum of all questions' maximum points. Since student $\mathcal{S}_1$ has answered all questions correctly his or her performance is 100% in all assessments.

Table 3.5.9: Normalization results

Scor	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$	$\mathcal{Q}_5$
$\mathcal{S}_1$	{2, correct}	{3, correct}	{4, correct}	{5, correct}	{6, correct}
$\mathcal{S}_2$	{2, correct}	{1, partially correct}	{4, correct}	{3, partially correct}	{6, correct}
$\mathcal{S}_3$	{2, correct}	{1, partially correct}	{2, partially correct}	{3, partially correct}	{0, wrong}
$\mathcal{S}_4$	{-2, wrong}	{1, partially correct}	{1, partially correct}	{1, partially correct}	{1, partially correct}
$\mathcal{S}_5$	{0, unanswered}	{-1, wrong}	{0, wrong}	{3, partially correct}	{-1, unanswered}
$\mathcal{A}_1$	•	—	—	•	—
$\mathcal{A}_2$	•	—	•	•	—
$\mathcal{A}_3$	•	•	•	•	•
Perf	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$
$\mathcal{A}_1$	$\{100\%, passed\} \equiv 7/7$	$\{71.4\%, passed\} \equiv 5/7$	$\{71.4\%, passed\} \equiv 5/7$	$\{0\%, failed\} \equiv -1/7$	$\{42.9\%, failed\} \equiv 3/7$
$\mathcal{A}_2$	$\{100\%, passed\} \equiv 11/11$	$\{54.5\%, failed\} \equiv 6/11$	$\{63.6\%, passed\} \equiv 7/11$	$\{9.1\%, failed\} \equiv 1/11$	$\{27.3\%, failed\} \equiv 3/11$
$\mathcal{A}_3$	$\{100\%, passed\} \equiv 20/20$	$\{70\%, passed\} \equiv 14/20$	$\{40\%, failed\} \equiv 8/20$	$\{15\%, failed\} \equiv 3/20$	$\{5\%, failed\} \equiv 1/20$

This intuitive approach works for most of the cases. In some cases however, additional normalization rules are applied. $\mathcal{S}_4$ achieved a negative accumulated score of -1 in $\mathcal{A}_1$. Since performance is usually interpreted as how much of a given task was successfully completed, a negative performance has no meaningful denotation. Furthermore, negative performance in one assessment could then count toward the final grade, effectively reducing performances in other assessments. Therefore, the default normalization rule is to clip performance to 0% in those cases—similar to the corresponding consolidation rule.

To derive the assessment result—that is, *passed* or *failed*—the passing threshold (see Section 3.2.2) is checked for each assessment. Another example of normalization rules are bonus questions, which lower the denominator of the sum of maximum points in the calculation of the performance value. Note that in this case the performance can possibly exceed 100%. Adjustments to individual students' performances represent a further class of normalization rules.

In our example neither bonus questions nor adjustments are applied. Thus, the assessment level grading is complete and we can proceed to the course level.

3.5.8 Step 6: Grading

Once all assessments of a course are finished the final grades are calculated.

Definition 3.5.8 (Grading)**.** *Grading is the process of combining performances* $Perf_{\mathcal{A}}(Scor, nR)$ *of all* $n \geq 1 \in \mathbb{N}$ *assessments* $\mathcal{A}_n$ *with weights* $w_{\mathcal{A}}$ *to generate an overall course performance. The course performance is then transformed to a grade value according to the grading scheme* gS*. Additional grading rules* gR *are applied afterward. The grading result is a pair, consisting of a grade value*

and a corresponding grading outcome, either passed or failed. The grading result is called grade $Grad_{\mathcal{C}}(Scor, gS, gR)$.

The grading results for the simulation are provided in Table 3.5.10. According to the definition above the grading process can be split into three steps:

1. Calculate an overall course performance.
2. Transform the performance value into a grade value.
3. Derive the grading result and apply grading rules.

Table 3.5.10: Grading results

$Perf$	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$
$\mathcal{A}_1$	$\{100\%, passed\} \equiv 7/7$	$\{71.4\%, passed\} \equiv 5/7$	$\{71.4\%, passed\} \equiv 5/7$	$\{0\%, failed\} \equiv -1/7$	$\{42.9\%, failed\} \equiv 3/7$
$\mathcal{A}_2$	$\{100\%, passed\} \equiv 11/11$	$\{54.5\%, failed\} \equiv 6/11$	$\{63.6\%, passed\} \equiv 7/11$	$\{9.1\%, failed\} \equiv 1/11$	$\{27.3\%, failed\} \equiv 3/11$
$\mathcal{A}_3$	$\{100\%, passed\} \equiv 20/20$	$\{70\%, passed\} \equiv 14/20$	$\{40\%, failed\} \equiv 8/20$	$\{15\%, failed\} \equiv 3/20$	$\{5\%, failed\} \equiv 1/20$
w	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$		
$\mathcal{C}_1$	20%	80%	—		
$\mathcal{C}_2$	20%	—	80%		
$Grad$	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$
$\mathcal{C}_1$	$\{6.00, passed\} \equiv 100\%$	$\{3.75, failed\} \equiv 57.9\%$	$\{4.25, passed\} \equiv 65.2\%$	$\{1.25, failed\} \equiv 7.3\%$	$\{2.25, failed\} \equiv 30.4\%$
$\mathcal{C}_2$	$\{6.00, passed\} \equiv 100\%$	$\{4.50, passed\} \equiv 70.3\%$	$\{3.25, failed\} \equiv 46.3\%$	$\{1.50, failed\} \equiv 12\%$	$\{1.50, failed\} \equiv 12.6\%$

The first step consists of calculating the weighted average of all single assessment performances in the course. The student $\mathcal{S}_2$ has passed the first assessment with a performance of 71.4%, failed the second with 54.4%, and passed the third assessment with 70%. For the first course only assessments $\mathcal{A}_1$ and $\mathcal{A}_2$ count. Since the second assessment has a higher weight, of 80%, the student's overall performance is only 57.9% (= 71.4%*20% + 54.4%*80%). Analogously, the overall performance in $\mathcal{C}_2$ is 70.3%.

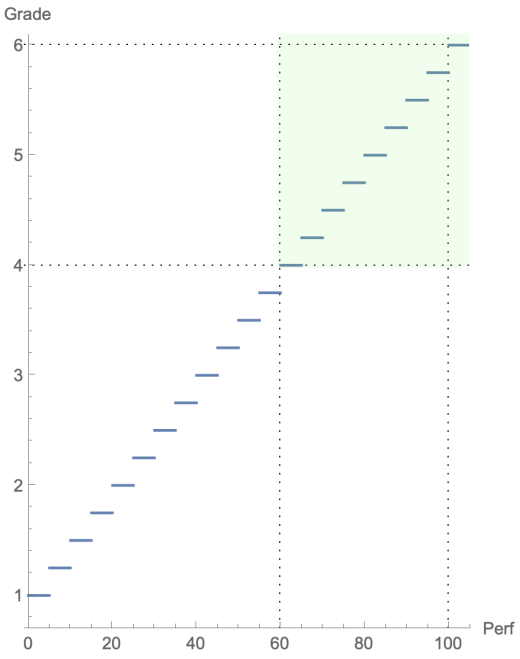
Using the overall performance, we can calculate the grade value in the second step. Depending on the country or educational organization, different grading schemes are applied. In our example we will use the Swiss university grading scheme.

Definition 3.5.9 (Grading scheme)**.** *A grading scheme* gS *is a list of* $g \geq 2 \in \mathbb{N}$ *grading associations. Each element is an association between a performance value and a grade value, a short signification, and a long signification, respectively.*

A grading scheme has the following properties: The interval of performances included in gS *is denoted by* $[P_{min}, P_{max}[$ *with* P_{min} *as the minimum performance and* P_{max} *as the maximum performance. Grade values are a sequence of* g *consecutive numbers or symbols ranging from* G_{min} *to* G_{max} *with constant increments of size* $incr$.

In Figure 3.5.2 we see the transformation of performances to grades for Swiss universities. The green area and the dotted lines indicate the passing

Figure 3.5.2: Swiss university grading scheme transformation



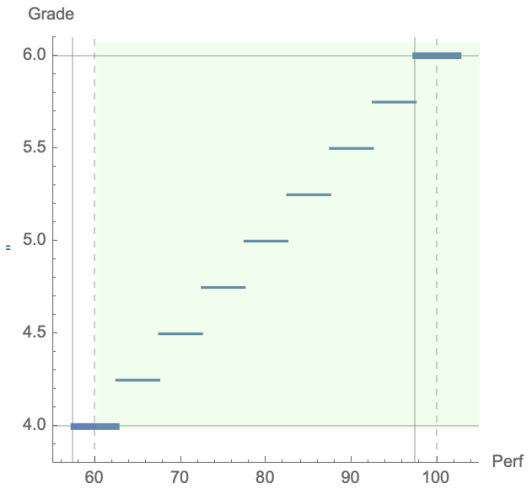
threshold—that is, the range of grades and performances which lead to passing a course. The transformation is a piecewise function since grade value increments are discrete; for example, there is no grade between 4.25 and 4.50. Thus, any grade covers an interval of performance values. The transformation function for $Grad(Perf, gS)$ is given by:

$$Min\left[P_{max}, \left\| \frac{1}{incr}\left(Perf\frac{G_{max}-G_{min}}{P_{max}-P_{min}} - \frac{G_{max}P_{min}-G_{min}P_{max}}{P_{max}-P_{min}}\right)\right\| incr\right].$$

Note that the performance has no upper bound (see previous section, 3.5.7), but grade values cannot exceed G_{max}. Since grades are defined as minimum performance requirements, any performance above the threshold of a particular grade will lead to the same grade unless the next highest threshold is reached. This leads to two practical problems: First, to achieve the highest grade *at least* 100% performance is required, which is a rare case in courses with several assessments, each containing of many questions. Second, in the student's perception it might feel unfair to get a lower grade when their performance is closer to the next higher grade value threshold.

To overcome these issues we can use a less strict grading procedure. If a student's performance is closer to the next highest grade value, we may round it up. Figure 3.5.3 shows the relevant part of the initial grading scheme (green area) compared to the rounding approach. The highlighted marks for the lower and upper bounds illustrate that with the rounded grading the student would

Figure 3.5.3: Rounded vs. strict grading



pass (4.0) already at a performance of 57.5%, subsequent grades follow with the same 5% increments, and the best grade (6.0) is awarded at 97.5%, accordingly. The transformation function is affected by this change only by changing the Floor function to Round.

While this grading approach proved to be less intrusive and gain higher acceptance among students, for our simulation we applied the regular (strict) grading[4]. Thus, for instance, student $\mathcal{S}_2$ failed $\mathcal{C}_1$ with a performance of 57.9%. In general we determine the grading outcome by simply checking the passing threshold for all students' grade values as shown in Table 3.5.10. In the grading scheme we have additional information for short and long significations (see also Figure 4.1.1). Those are used in the reporting of the evaluation results to the student. However, they are already associated with the grade value and not modified in the grading process. Therefore, the significations don't need to be included in the grading results.

Again, at this stage additional grading rules are applied, in case exceptions need to be handled. Since we don't have any such cases in our simulation we proceed with the final step of certification.

3.5.9 Step 7: Certification

The certification step is not yet fully implemented in SYLVA since it has only been used for single courses so far. In order to give a comprehensive overview of the grading process it is explained here. Conceptually, the certification is similar

[4]In SYLVA different grading schemes for several countries and school systems are implemented and educators can switch between them conveniently without the need to transform or recalculate grades.

to the grading step; that is, lower level results are combined and condensed in a single outcome.

Definition 3.5.10 (Certification)**.** *Certification is the process of calculating average grade values and assigning degrees to students based on the* $c \geq 1 \in \mathbb{N}$ *grading results* $Grad_{\mathcal{C}}(Scor, gS, gR)$ *according to the certification scheme* cS*. The grading result is a pair, consisting of a grade value average and a certification outcome, either cert or empty, i.e., no certificate. The certification result is called certificate* $Cert_{\mathcal{P}}(Grad, cS)$*.*

Table 3.5.11: Certification results

$Grad$	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$
$\mathcal{C}_1$	$\{6.00, passed\} \equiv 100\%$	$\{3.75, failed\} \equiv 57.9\%$	$\{4.25, passed\} \equiv 65.2\%$	$\{1.25, failed\} \equiv 7.3\%$	$\{2.25, failed\} \equiv 30.4\%$
$\mathcal{C}_2$	$\{6.00, passed\} \equiv 100\%$	$\{4.50, passed\} \equiv 70.3\%$	$\{3.25, failed\} \equiv 46.3\%$	$\{1.50, failed\} \equiv 12\%$	$\{1.50, failed\} \equiv 12.6\%$
$Cert$	$\mathcal{P}$				
$\mathcal{S}_1$	$\{6.00, Cert\}$				
$\mathcal{S}_2$	$\{4.13, Cert\}$				
$\mathcal{S}_3$	$\{3.75, —\}$				
$\mathcal{S}_4$	$\{1.38, —\}$				
$\mathcal{S}_5$	$\{2.50, —\}$				

The certification scheme is, analogous to grading schemes, a list of associations between aggregated grade values and significations. Typically at this level only a few Latin honors, such as *rite*, *magna cum laude*, and *summa cum laude* are distinguished. The corresponding grade values required for qualifications vary between institutions and countries. The actual certificate is awarded based on a threshold, in our example a minimum grade value of 4.0. Additional requirements may apply depending on the institution and the study program. In this step no transformations are needed and usually no exceptions are handled. In Table 3.5.11 the certification results are presented. While student $\mathcal{S}_1$ obtained the certificate with the best possible average grade, students $\mathcal{S}_4$ and $\mathcal{S}_5$ failed both courses and will therefore not get a certificate. Students $\mathcal{S}_2$ and $\mathcal{S}_3$ both passed and failed one course, respectively. However, only student $\mathcal{S}_2$ is eligible for the certificate with an average grade above 4.00. In contrast to course level grades, on the study program level average grades are usually not rounded to the grade increments.

3.6 Reporting

Starting with a discussion of question and answer types, we came up with five example questions for which we calculated the grading in the previous sections. Now, we'll cover the reporting of the evaluation results by deriving report components from abstract evaluation aspects. These components are used to create the reports in SYLVA.

3.6.1 Criticism, evaluation, and ranking

According to Wolff (1969), reporting (of evaluation outcomes) consists of three main aspects—criticism, evaluation, and ranking. He defines "the three species of grading" as follows:

- Criticism
  "[T]he analysis of a product or performance for the purpose of identifying and correcting its faults or reinforcing its excellences"

- Evaluation
  "[T]he measuring of a product or performance against an independent objective standard of excellence"

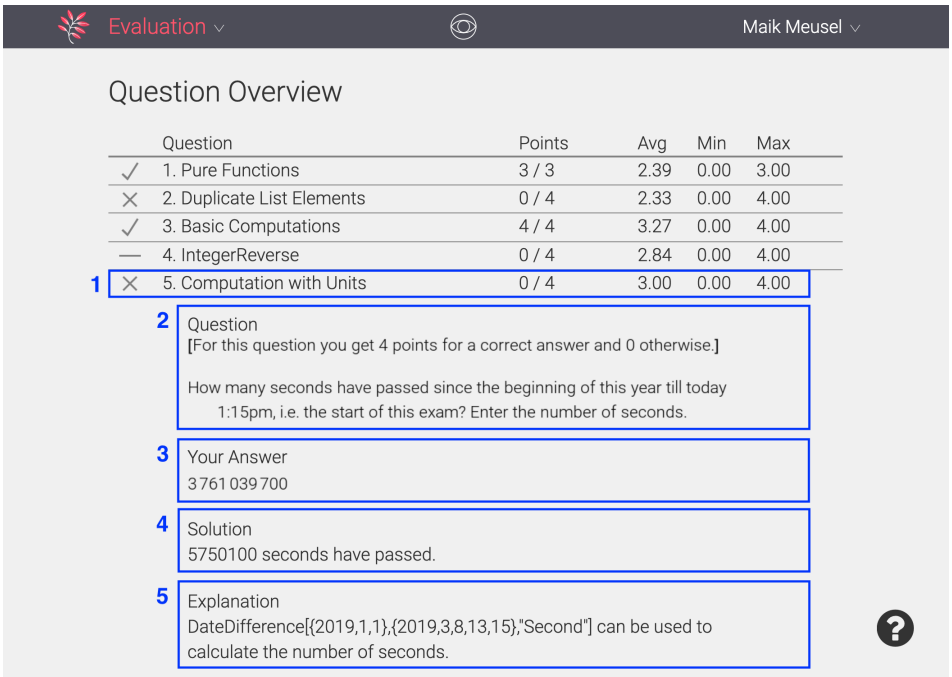
- Ranking
  "[A] relative comparison of the performances of a number of students, for the purpose of determining a linear ordering of comparative excellence"

These definitions—while conceptually appropriate—were developed long before computers were used for assessment and evaluation. Thus, we need to interpret them with regard to the opportunities a modern education platform provides.

In order to incorporate these concepts, we make use of the intermediate results from the seven-step-grading process. To analyze the performance for criticism, for each answer we need to extract and display the (1) assessment outcome class (here denoted by the "X" symbol), (2) the initial question text (and dataset, if applicable), (3) the answer given by the student, (4) the solution, and (5) the explanation, as shown in Figure 3.6.1. All questions in the list are displayed as a compacted overview, and details expand once the student clicks on the question. At first, therefore, one sees the outcome class, which provides a quick indicator whether the question was answered correctly. Typically, a student would be less interested in details about a question he or she answered correctly. Note that the outcome class itself does not tell *what* was wrong, and is therefore not sufficient for drawing meaningful conclusions—that is, for learning from the error. Only by confronting the answer given and (one of) the solution(s)—along with the explanation—does the student get the additional information necessary to understand his or her error.

The measuring of performance against an objective standard can—for our purposes—be interpreted as calculating the total of points scored (performance) and bringing it into relation with the maximum points (the standard of excellence). This can be achieved with a list (as shown in Figure 3.6.1) of (score) numbers that are summed up against the maximum points. On an aggregated level, the performance (for assessments) can simply be shown as a single number since it is common sense that performance is measured in percentage terms, thus normalized. The same logic applies on the course level, where the performance

Figure 3.6.1: Question details



is aggregated and transformed into a grade. Note that comparing a particular performance to the passing threshold, or to the average performance, can be interpreted as an independent standard of excellence too.

Ranking of one student relative to the others can be incorporated by means of comparison with the average performance, by defining performance classes and assigning each student to such a class, or by an ordered list of performances—either numerical or graphically. All these different means of comparison can be—due to automated computations—provided effortlessly. At this point, therefore, we will not discuss which of them is the most meaningful but assume they all provide useful insights—depending on the context or perspective.

3.6.2 Report components

In this section we build report components based on the previous discussion. These components, as summarized in Table 3.6.2, can be flexibly combined to form reports for different audiences (students and educators), and contexts (assessment and course reporting).

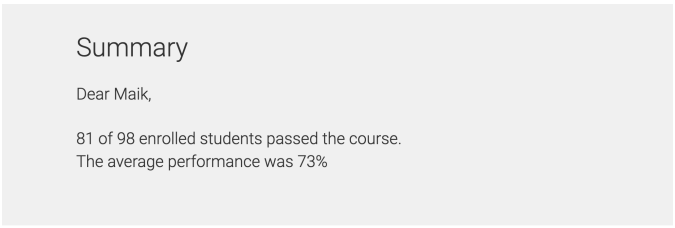
SYLVA reports contain the following components[5]:

- Performance bar

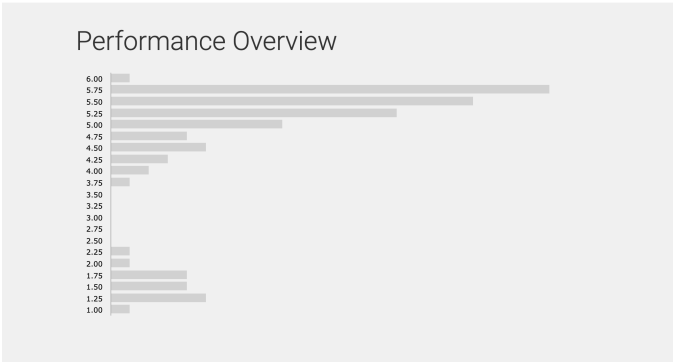
[5]Since the figures of the components that are shown in the overview only occur in the context of reports, these figures are not labeled seperately or referenced to individually.

Shows a condensed measure—either performance or grade—to summarize the overall evaluation outcome.

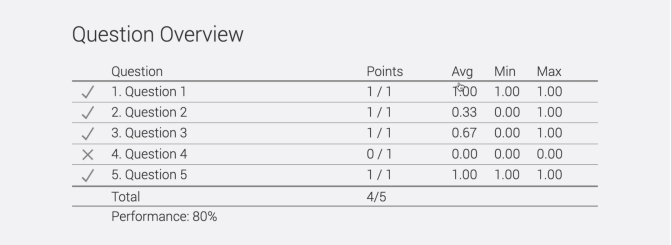
- Summary
  Summarizes the evaluation outcome verbally by providing information about success, a classification of performance, and a relative comparison, along with a motivational or concluding statement.



- Performance overview
  Graph that shows how many students fall into particular performance classes.

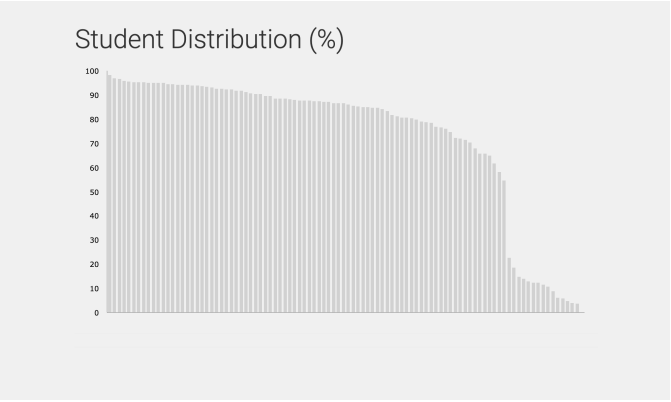


- Question overview
  List that, for each question in an assessment, shows the assessment outcome (class), the points scored, in comparison to the maximum and minimum points, along with the average score.

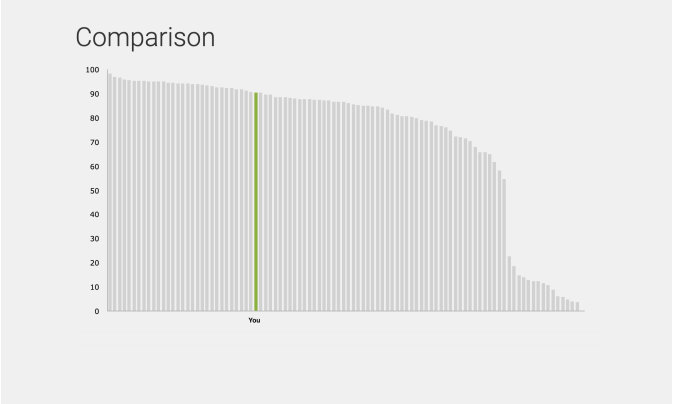


- Assessment overview
  List that, for each assessment in a course, shows the weight and performance—either individual or average.

- Student distribution
  Graph that shows the ordered linear distribution of all single performances.

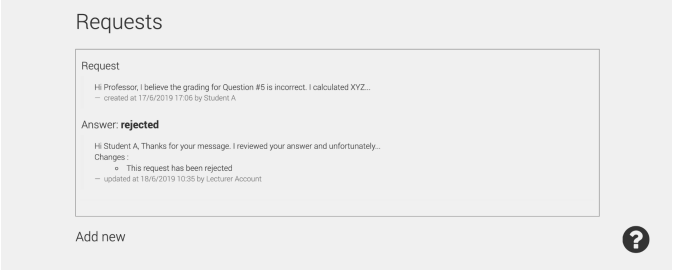


- Comparison
  Student distribution with the student's performance highlighted.



- Student list
  List of all students with their respective performances or grades.

- Requests
  UI element to file requests. Any requests or grading adjustments are displayed below.



While most of the components are self-explanatory with regard to the evaluation aspect they incorporate, the summary—as an attempt to verbalize the entire feedback in a condensed form—targets all three aspects. In the current implementation, this is based on a simple decision tree and cannot be customized or individualized further by the educator. This brings with it the advantage that the reporting can be fully automated and therefore requires no additional effort of the educators. Admittedly, however, such template-like feedback will feel jaded once a student has taken several courses, and may not adequately represent the "lessons learned" educators wanted to teach.

Table 3.6.1: Report components overview

	Criticism	Evaluation	Ranking
Performance bar	•	•	—
Summary	•	•	•
Performance overview	—	—	•
Question overview	•	•	—
Assessment overview	—	•	—
Student distribution	—	—	•
Comparison	—	—	•
Student list	—	—	•
Requests	—	—	—

3.6.3 Report types

Since, so far, SYLVA is used for single courses only, for the reporting we distinguish four types of reports:

- Student assessment report
  Individualized report for a single student with regard to a single assessment.
- Educator assessment report
  Aggregated report for educators, containing all results of a single assessment.
- Student final grading report
  Individualized report for a single student with regard to a single course.
- Educator final grading report
  Aggregated report for educators, containing all results of a single course.

Reports on the study program level—similar to those on the course level—can be added once the platform has achieved greater maturity. The distinction between a student version and a educator version is necessary to comply with data privacy norms—so, for example, no student should see the identifiable results of any other student. This information, however, is necessary if educators are to target their attention and help toward specific students. This is the reason why we have components—for instance, the Student list—that are only available to educators. Note that educators can navigate to all student reports in order to not have to show educator reports to students during discussions.

Table 3.6.2 compares the four report types with regard to their components. As can be seen, for all report types there is at least one component that covers each of the three evaluation aspects. The only report component that is unrelated to these aspects is the *Request* feature. Note that it is placed at the

Table 3.6.2: Report types overview

	Assessment report		Final grading report	
	Student	*Educator*	*Student*	*Educator*
Performance bar	•	•	•	•
Summary	•	•	•	•
Performance overview	—	•	—	•
Question overview	•	•	•	•
Assessment overview	—	—	•	•
Student distribution	—	•	—	•
Comparison	•	—	•	—
Student list	—	•	—	•
Requests	•	—	•	—

end of every student report for convenience purposes, and to make sure requests stay well-organized for educators. In Section 4.4 we discuss how requests are handled.

Chapter 4

Administration

Since we have already covered many of the aspects of administrating courses, in this chapter we will discuss only a few additional topics. While basic administration features such as inviting users and exporting grades are common to most educational platforms, an ILAP—due to its deep level of integration—offers additional opportunities to analyze courses and manage assessments and evaluations.

In this chapter, therefore, we discuss how projects and organizations are initially created; then, how teams and presentations are assigned with regard to students in a course. In the subsequent sections we explain the ILAP specific features around course analytics and reviewing grading and student requests.

4.1 Project and organization setup

While users can create new courses and invite additional collaborators once they are on the platform, initial setups are needed for this to function. SYLVA has a separate administration tool (see Figure 4.1.1) for such purposes and system administration. This tool offers the following features:

- Projects
  Set up, view, edit, and delete projects.

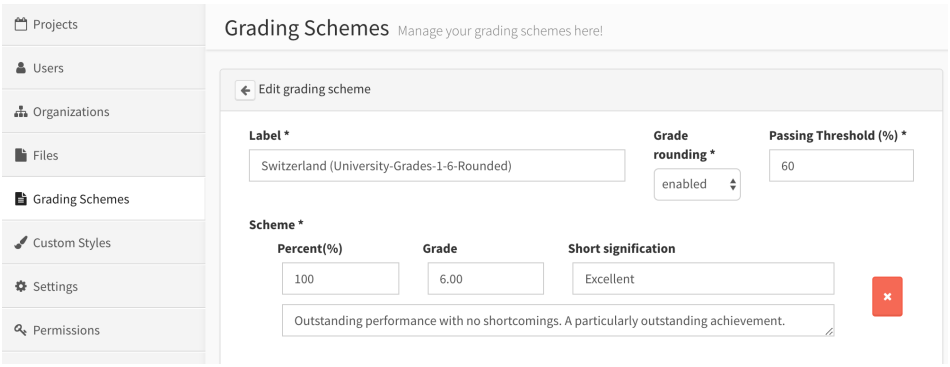
- Users
  Invite, create, view, and edit users. Assign users to projects and organizations. Set and reset roles and passwords.

- Organizations
  Create, view, edit, and delete organizations.

- Files
  Upload, view, download, and delete files of all projects.

- Grading Schemes
  Create, view, edit, and delete grading schemes.
- Custom Styles
  Create, view, edit, deploy, and delete CSS style themes.
- Settings
  Create, view, edit, and delete various configuration files.
- Permissions
  View roles and permissions. View all related API methods. Simulate various actions via an API explorer.

Figure 4.1.1: Editing grading schemes in the administration tool



To maintain and administer the platform there is a significant overlap of features in the administration tool and the Assessment app. Most of the features from *Projects* can be used by regular users with sufficient permissions. However, to prevent misuse of identities and preserve unambiguity, currently only platform administrators can create institutions in the *Organizations* menu. All projects and users need to be associated with organizations. Styles, logos, and grading schemes are also inherited from the organization. We already discussed grading schemes in Section 3.5.8. To attain consistency across the organization they can only be created and edited from the *Grading Schemes* menu by administrators.

The last three items, *Custom Styles*, *Settings*, and *Permissions* are entirely for technical maintenance purposes and therefore not relevant to users of the platform.

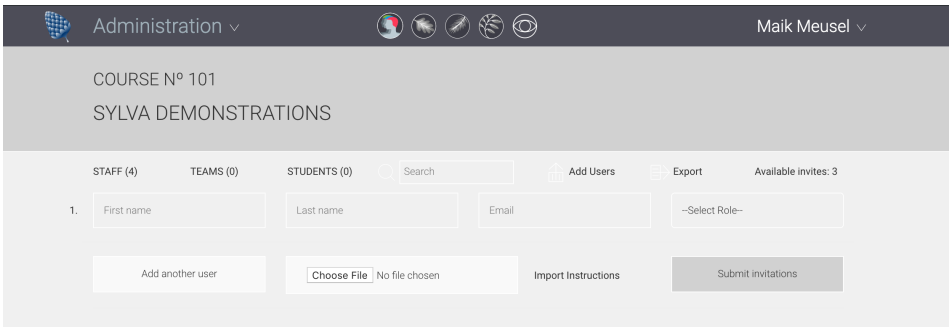
4.2 Managing users, teams, and presentations

In this section we will discuss how to manage larger numbers of students—assigning them to teams or presentations and, in turn, assigning teams themselves to presentations.

Bulk invitations, seats, and exporting users

In Section 2.3.1 we discussed the invitation of single users. Figure 4.2.1 shows the invitation UI with additional invitation features.

Figure 4.2.1: Inviting users



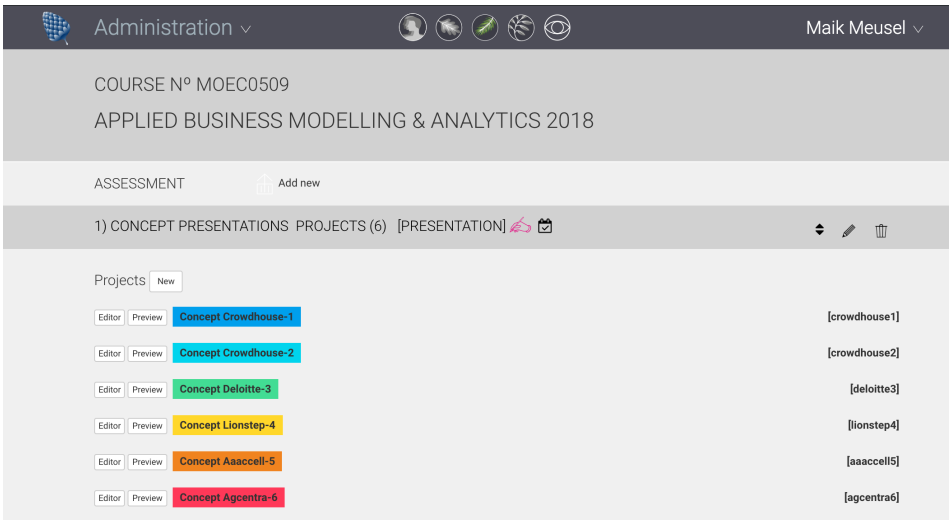
To invite several user simultaneously, the bulk invitation feature can be used to upload CSV files that contain the following data: *first name*, *last name*, *email*, *role*. This allows one to set up entire courses, including staff roles, from a single list. The list of the current users can be exported at any time using the *Export* button shown in Figure 4.2.1. In the right corner of the figure, *Available invites* shows the number of free seats that are allocated to the course. Seats are used to limit the access to a course for monetizing purposes. Educators or administrators can buy seats in order to invite students to courses. There are additional features related to the payment and transferring of seats, but these are not explained in further detail here.

Teams and presentations

To conduct team assessments the students of a course must be assigned to groups. First, teams have to be created from the users tab in the Administration app. Students can then be automatically distributed among teams, or the educator can assign students to teams individually. It is possible to create more teams than students and each student can be a member of more than one team. While often teams stay constant during the course, the flexibility in team creation allows one to form new teams for different assessments.

Once the students are distributed among teams, the educator only needs to switch from individual to team assessment. All the students of a particular team share the same assessment (specimen), including possible parametrization and randomization of questions. This means each student can access the assessment and modify the answers or slides (for presentations). Here, any submitted answer will overwrite the previous answer, regardless of who submitted that answer initially.

Figure 4.2.2: Setting up team presentations



All team members will get the same evaluation results and reports. Since the assessments are directly linked to the teams, students should not be reassigned once an assessment has started. Removing students from a team will not only disable their access but also delete any previous evaluation result that was attained by the group. Therefore, additional teams have to be set up in order to redistribute students after an assessment has finished.

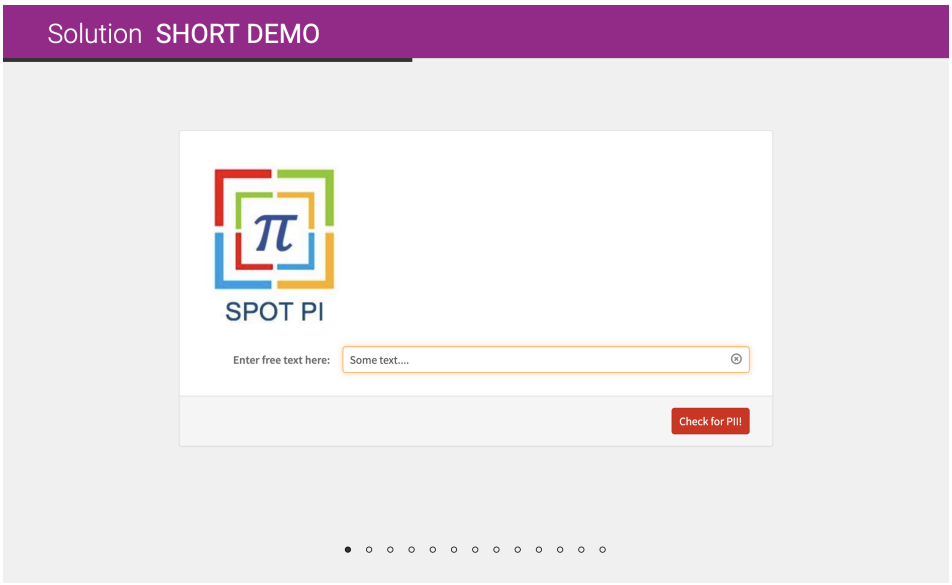
For seminars, presentations or term projects are usually the main assessments. Presentations can be set up with the assessment type *Presentation* in the Administration app. Once the grading rules and structure of the project are set, specimens can be created for each team, as shown in Figure 4.2.2. These are different versions of the presentation that share the same rules and schedule. Beside the title of the presentation, the background image and colors can be adjusted.

For presentations, unique public links are generated so that they can be embedded or collected in separate web pages, for instance the student presentation shown in Figure 4.2.3, which is hosted on www.abma.ac/2019.

4.3 Course analytics

Tracking students' learning and assessment behavior along with their evaluation results allows us to gain insights on how courses can be improved and how student success can be increased. By comparing the time students spend on viewing the materials and answering questions we get indicators that help us to identify (too) difficult questions or chapters in the courseware. On the individual student level comparing activity can help us to identify the struggles of students

Figure 4.2.3: Example of a standalone student presentation



who need more support in order to succeed.

These ambitions are part of ongoing research projects. At this stage, no features able to deliver such insights have been implemented. Note that only an ILAP can extract behavioral data on the student level and thus lead to such a comprehensive view of student success in relation to a student's efforts. Thus, little data and research exist in this area.

In the current implementation of SYLVA the following data are created automatically:

- Pageview records (optional)
  Contains, for each student, all pages visited with corresponding time stamps.
- Lecture analytics (optional)
  Contains, for each student, accumulated time spent in seconds for all chapters of all lectures.
- Grades overview (compulsory)
  Contains, for each student, the grade, the overall performance, and the performance in each assessment.
- Assessment questions (compulsory)
  Contains, for all assessments, all questions with corresponding average, minimum, and maximum points.
- Assessment evaluation (optional)
  Contains, for each student and each assessment, the start and submission

Figure 4.3.1: Assessment analytics data

Programming Bootcamp 2019_assessmentQuestions

assessment	question	average	min	max
Assignment 1	Map and Apply for Lists	1.836734693877550	0	2
Assignment 1	Properties of Lists	1.9387755102040800	0	2
Assignment 1	Wolfram Language Properties I	1.3673469387755100	1	2
Assignment 1	Wolfram Language Properties II	1.9591836734693900	0	2
Assignment 1	Wolfram Language Functions I	1.836734693877550	0	2
Assignment 1	Problem Solving I	1.5408163265306100	1	2
Assignment 1	Problem Solving II	1.6734693877551000	0	2
Assignment 1	Problem Solving III	1.4285714285714300	0	2
Assignment 1	Transposition of Lists	1.9591836734693900	0	2
Assignment 1	Write Code I	3.795918367346940	0	4
Assignment 1	Write Code II	3.448979591836740	0	4
Assignment 2	Units & Formats 1	1.8775510204081600	0	2
Assignment 2	Units & Formats 2	1.6938775510204100	0	2
Assignment 2	Wolfram Language Properties	1.1020408163265300	0	2
Assignment 2	Working with Data 1	3.551020408163270	0	4
Assignment 2	Working with Data 2	3.3061224489795900	0	4

time and points scored in each question.

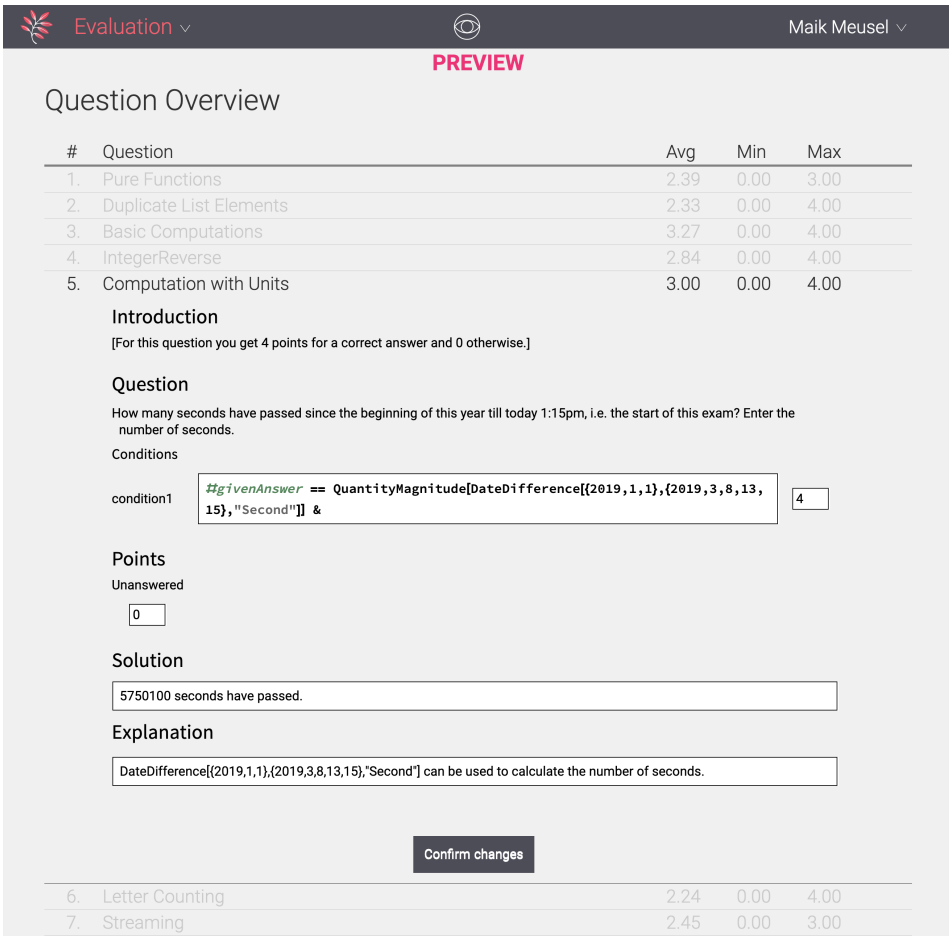
- Assessment analytics (optional)
  Contains, for each student and each assessment, the time spent on each question.

In Figure 4.3.1 an example of lecture analytics data is given. All datasets are currently raw data that can be downloaded in CSV format from the Evaluation tab in the Administration app. The datasets can be categorized into *optional* and *compulsory* data. While all data are sensitive and private to some extent, evaluation results data are, in particular, key in terms of the main purpose of the platform—that is to say, generating grades—and tracking therefore cannot be disabled for these types of data. The tracking of optional data, however, can be turned off by the educator to avoid privacy violations.

4.4 Evaluation and requests

Following up on reporting, discussed in Section 3.6, we now cover the handling of requests in the Administration app in the present section. Reviewing requests is an important part of educators' daily business as it contributes—positively or negatively—to student satisfaction. In SYLVA, educators can act directly upon requests, and adjust grading—for individual students or the entire class. We first explain the reviewing and publishing of grading, before moving on to a discussion of how adjustments *can*, and how they *should*, be carried out.

Figure 4.4.1: Modifying conditions and scoring rules



Reviewing and publish grading

Once grading has been triggered by the educator, he or she sees a preview of the evaluation results. These results are unpublished by default so that the educator can check the plausibility of the results and adjust the grading—if necessary.

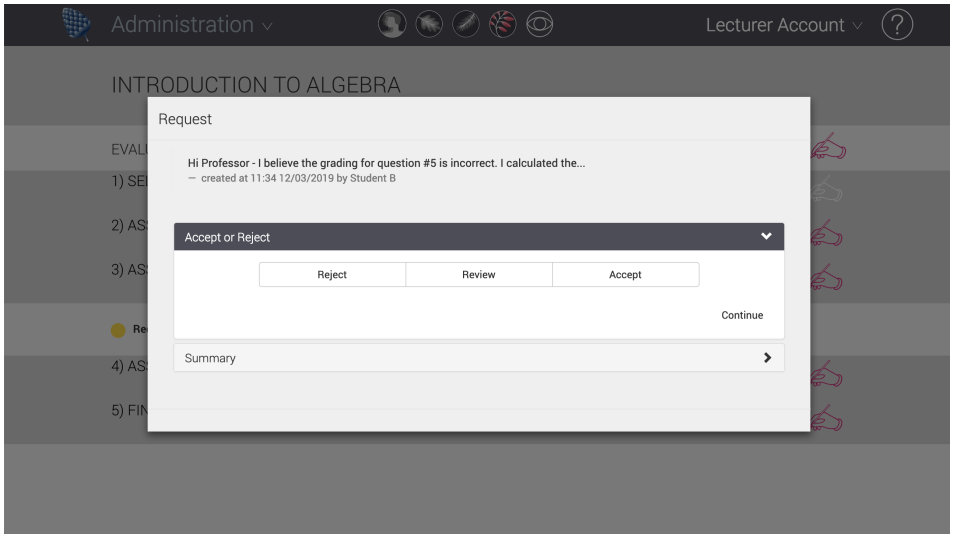
In some cases the statistics provide insights that can be used to detect errors in the grading. For instance, if none of the students in a large course scored points in a particular question, chances are that either the question was misleading or the conditions or scoring rules weren't set up properly.

In Figure 4.4.1 we see where the scoring and inquiry conditions can be modified. The UI replicates some of the functionality of the CREO assessment notebook templates. Not only is editing grading directly after its review more convenient for educators, it is necessary, because changes that are made in CREO will not affect an assessment that was deployed, retrospectively. In fact,

CREO is designed as a one-way deployment tool. In the modify grading UI, of course, it is no longer possible to change a question, but the solution and explanation can be modified to adopt changes to the inquiry conditions or scoring rules.

After the changes are submitted, all students' answers are regraded and a new preview is presented to the educator. Once the educator has made all the necessary adjustments, the results can be released to the students. Note that modifications to the grading can be made at any time after the results are published—then, however, all modifications are tracked and displayed to students. To avoid unnecessary requests, it is advisable to correct obvious errors before the initial publication of the results. The same logic applies to the final grading.

Figure 4.4.2: Handling requests



Reviewing student requests

Once the evaluation results are published, students can review their reports and file requests if they disagree with how particular questions were graded. Note that the solution and explanation should ideally provide enough information for the student to understand the evaluation, so that requests are used only to handle disagreements rather than general questions. Educators see incoming requests associated with particular assessments, or the final grading, in the Evaluation tab of the Administration app. In Figure 4.4.2 we see that for each request, educators can chose between the following options:

- Reject

The request is returned without any changes to the grading. Optionally, a justification can be added by the educator.

- Review
  The request is returned with an explanation of how the grading was adjusted, or why it was not adjusted.

- Accept
  Based on the request, any of the following adjustments—optionally with an additional explanation—is applied:

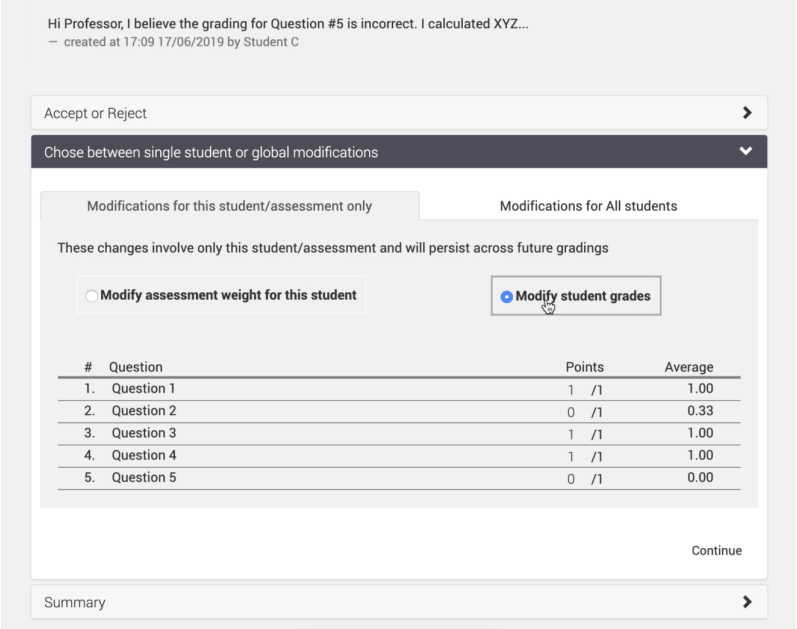
  - Change grading conditions for the assessment
  - Change scoring for the student in question
  - Change scoring for all students
  - Change assessment weight for the student in question
  - Change assessment weight for all students

While *rejecting* a request does not require additional action by the educator, it is reasonable to add an explanation, for the student, as to why the request could not be accepted. The *reviewed* requests are similar to *rejected* requests in the sense that no adjustments are undertaken by the educator. The reason for this distinction is, therefore, for communication purposes only. If there was an obvious error in the grading, many students will file the same type of request asking for the same adjustment. The educator can adjust the grading for all students while handling the first of these similar requests. Once the adjustments have taken effect, all other requests are obsolete. But, in order that students understand that their requests have been addressed, this option can be used and, thus, requests are not marked as *rejected*. The same applies to cases where the request is not clear enough for the educator to act on it. Note that requests are not comparable to chats. For each request only one reaction is allowed, and students cannot reply to that action, but must file a separate request if necessary.

Only *accepting* a request leads to changes in the grading. Here, educators can chose between adjustments on the individual student level and those that apply to all the students. In general, modifications that only apply to a single student should be avoided since they can, potentially, violate fairness norms. Thus, educators should always consider changing the conditions, scoring, or assessment weights for all the students before considering individual modifications.

Typically, conditions are adjusted when it turns out that a particular question was graded falsely or some valid answers were not covered. If this is not possible, the scoring can be adjusted to either change the relative impact of a particular question or to change the allocation of points with regard to the four outcome classes—that is, how many points are awarded for correct, partially correct, or wrong answers, and the case in which the question was not

Figure 4.4.3: Modifying individual scores



answered. In some cases the question turns out to be indeterminate or simply misleading. Then, the best adjustment is to award the same amount of points to all answer classes, since it is not clear why students did (not) give a particular answer to the question. When the maximum points for a question are changed, the relative impact—compared to other questions—will change. This can be done in cases, for instance, where a question turns out to be too difficult. When scoring adjustments are not sufficient, the last measure is to change the assessment weight. This will affect the relative impact of the particular assessment with regard to the final grade. In most cases, this is used to make an assessment no longer count—that is, to set the weight to zero.

Changing the grading on the individual student level, as shown in Figure 4.4.3, is only necessary in a few exceptional cases. By way of an example, consider a coding assignment where the evaluation time or memory of a particular code is to be analyzed. Depending on technical issues (related to the operating system, other software, or hardware), the particular code may not evaluate correctly on a student's computer, and either give no, or distorted results. Then, the student cannot solve the assessment successfully because he or she will come up with wrong, or no answers at all. In this case, adjusting the conditions is not possible without introducing complete arbitrariness. Thus, an exceptional adjustment of the individual's scoring (if only a few questions are affected) or the assessment weight (if the entire assessment is affected) is the best approach to dealing with the problem.

Chapter 5

Technical design

In this chapter we will discuss a few specific topics regarding the technical architecture of SYLVA. Modern web platforms are usually designed to be scalable to serve large numbers of user across many countries, and flexible with regard to the integration of services from other providers, or integretaion into other platforms. To attain these properties, it is becoming more and more popular to use DevOps, microservices, API functions, and serverless functions. While SYLVA incorporates all of the aforementioned concepts and technologies, the focus in this chapter is on the specific computational needs for ILAPs. In the first section we will look at how the apps are structured and what implications this has for the navigation. We will then discuss how the three main types of computations are handled by the back end, before we explain in more detail the use of the WEPC for scientific computations. Lastly, we will cover the main technical authorization and security concepts that are used in SYLVA.

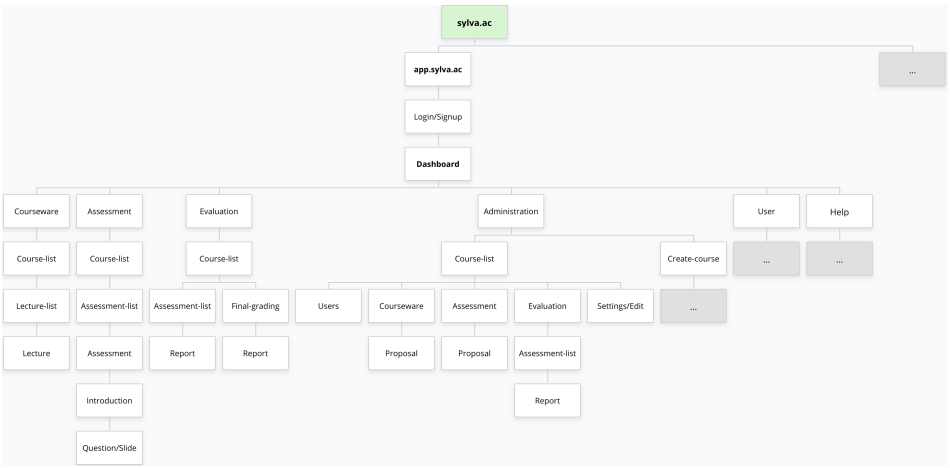
5.1 Platform navigation

In this section we will look at platform navigation to get an understanding of the structure of and connections between the features, which are organized in apps. Most educational tools and platforms appear to be very complex, confusing, and overloaded. Therefore, the main design targets for SYLVA are simplicity, consistency, and transparency.

In contrast to static content, on SYLVA all materials can contain interactive elements. Therefore, the platform is designed to be simple—that is, as little intrusive as possible—by minimizing the number of navigational components and keeping their positions fixed. To reduce the learning effort required by specific workflows, apps and features are designed to be consistent and as similar as possible, as can be seen in Figure 5.1.1. Transparency is achieved by offering the same user experience and content to all users.

On the landing page, sylva.ac, visitors first get basic product information

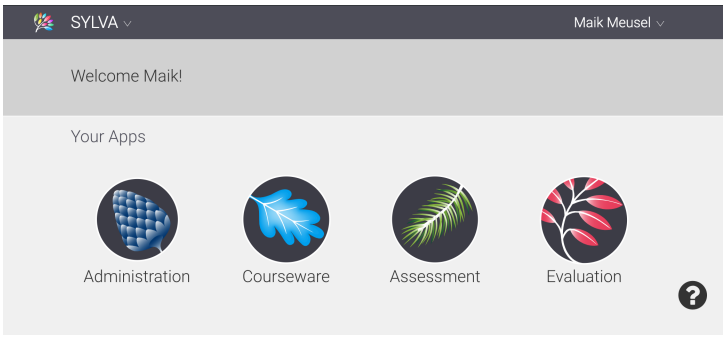
Figure 5.1.1: Platform sitemap



and can sign up for an account on the platform. Once they are registered, they can enter the platform via app.sylva.ac. Following login, every user sees the dashboard. The dashboard is the center for platform navigation. From there, all apps and features are accessible, as shown in Figure 5.1.2.

Beside the four core apps we discussed in detail in recent chapters, only the user page and the help page are added to platform. The dark gray area at the top of the page is the fixed navigation bar. For consistency, it remains in place as the user navigates to other pages and has the role of providing an anchor point for orientation. On the left side of the page the current app is displayed in the navigation bar, and from there users can switch to other apps. From there, users can switch to other apps. The user menu page is accessed by clicking on the name on the right side of the navigation bar. In this menu typical platform features such as log out and the user profile, which allows a user to change passwords and settings, are grouped together.

Figure 5.1.2: Platform dashboard



Help features are accessible via the question mark button at the bottom right of the page. Here users can ask questions and get access to other support resources. Like the navigation bar, the help button is fixed and is visible on all pages in SYLVA.

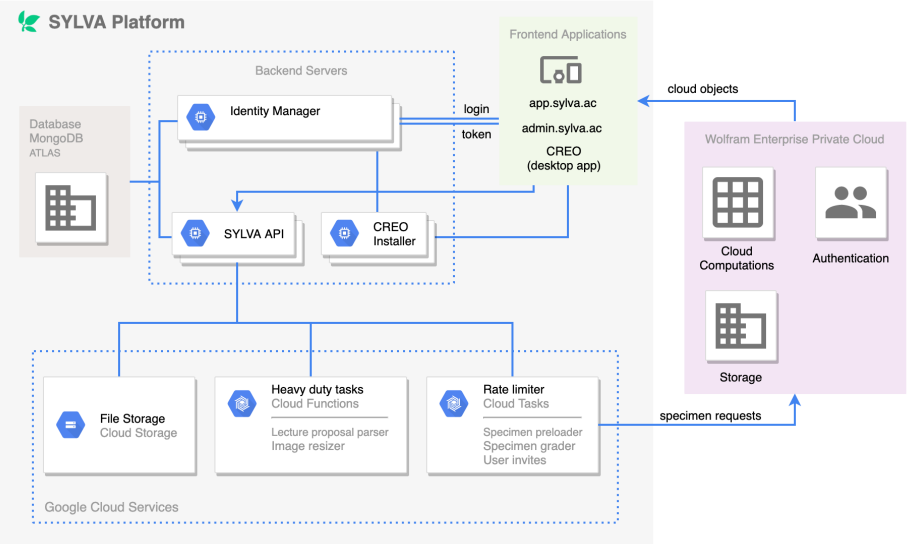
The structure of the core app navigation is consistent across all apps. Upon starting each app the user has to select a course from the course list page. Then, the user sees the components list, from which all components are directly available. For the Courseware app this means the list of lectures (Figure 2.4.1); for the Assessment app it means the list of assessments; in the Evaluation app, the list of corresponding reports and the final grading. The Administration app has additional tabs for creating and editing of courses, user management, and one each for the three other core apps. Note that this navigation structure is the same for all users regardless of their role.

At this point one may wonder why the Administration app has tabs for the core apps, or why the administration features are separated from the other apps. While this overlap adds complexity to the navigation, it is necessary to attain the aforementioned consistency and transparency goals. There are two main reasons for this design: First, users can have different roles in different courses. A teaching assistant who creates materials and assessments in one course can be a student in another course. Therefore, different versions of each app would be needed to distinguish the features and permissions associated with different roles. Second, educators are using the same apps as students for teaching (in-class presentation). If the editing and administration features were combined in the same apps, students could see some details of the authoring and administration process, including for instance who created a certain lecture and when the proposals were accepted. In addition to the fact that some educators may be opposed to this, it brings with it the risk of unintentionally revealing certain details of assessments or private information about student evaluations. While having a teaching platform that integrates authoring and administration features brings many benefits for educators, at the same time it makes sense to draw a clear line between in-class usage on the one hand and preparation and administration on the other. The current design maximizes transparency and consistency because students and educators use the exact same apps with the exact same content and features.

5.2 Back end architecture

In this section we will discuss the technical architecture of SYLVA with a focus on the computational requirements. Figure 5.2.1 provides an overview of the main back end components. The service providers mentioned here are those for the current implementation of the platform. The architecture can, however—without loss of generality—be implemented with alternative providers that offer comparable features.

Figure 5.2.1: Computational architecture



To enable the services and operations of the platform the back end needs to be capable of providing three main classes of computations:

- Time-sensitive computations
  Usually small but frequent computations mainly used for authentication and front end interactions.

- Heavy load computations
  Memory or CPU intensive computations mainly used for deployment of content.

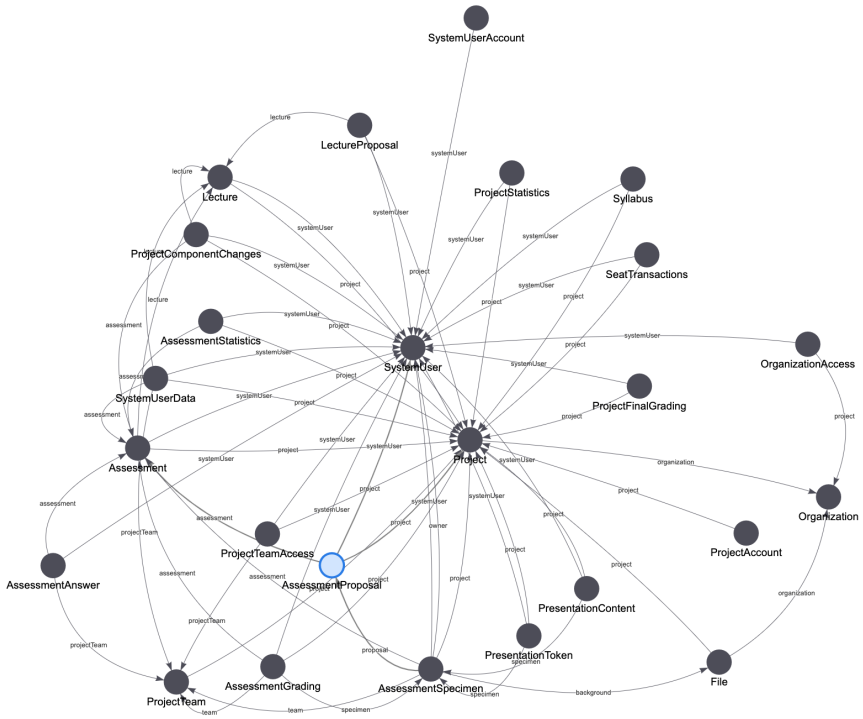
- Specific mathematical computations
  Scientific computations mainly used for grading, creation, deployment, and provision of interactive elements in the course materials and assessments.

While the first two classes are typical of most web applications, the last is special with regard to the academic content the platform is designed for. Therefore, we will cover the specific mathematical computations performed by the WEPC in a separate section, 5.3.

Time-sensitive computations involve login and any actions that require authentication and are handled by the identity manager. The services provided by SYLVA are organized into several APIs, including—for example—the Assessment app. Since almost all user interactions with these services require authentication, the identity manager has to process high volumes of small requests. For this purpose fast read and write access to data, which is achieved

by the database connection, is essential for a smooth user experience of the core applications. In SYLVA most actions involve individual, user-specific data that need to be separated. Thus, the database schemas are highly complex and require a lot of flexibility when additional features are developed iteratively. An overview of the 25 schemas (each of them has up to 30 properties) of the current implementation is given in Figure 5.2.2. To attain the agility necessary to extend and optimize the schemas, NoSQL databases such as MongoDB[1] are well suited. With MongoDB Atlas[2] the database is operated as a cloud service.

Figure 5.2.2: Data schemas



The class of heavy load computations first of all contains the parsing of lectures (see Section 2.2), and the provision of assessments. The former process requires the creation of XML files to deploy the HTML pages. For each interactive element a separate CloudObject is created (see Table 2.2.2). For all static graphics SVG files are created. Depending on the lecture this can take more than a minute and create dozens of files. In addition, course content created in CREO is stored as Mathematica notebook files, which can have a size of several megabytes. Since lecture deployment is not time-sensitive more cost-efficient

[1] MongoDB is a document database (MongoDB, 2019b).

[2] MongoDB Atlas is a database as a service created by the experts who design and engineer MongoDB (MongoDB, 2019a).

options such as object storage or file storage are sufficient. SYLVA uses Google Cloud Storage to store files from CREO.

Assessments are due to the authoring environment, comparatively standardized and vary less in size. In contrast to courseware they are always individualized—that is, separate specimens are created for each student, and can entail parameterization and randomization. For this reason it is useful to save them as templates in the JSON format. While template storage consumes only little space, each specimen contains several CloudObjects (at least as many as the number of questions), which need to be created. Depending on the class size and the type and number of questions this can take several minutes in total.

In Section 3.2.1 we discussed how waiting times for students starting exams are minimized. Nevertheless, these tasks remain technically challenging because they involve many requests being sent almost simultaneously. For repetitive tasks like specimen creation and grading or bulk invitations rate limiters are used to protect the computational instances from overflows, which could lead to crashes. This applies in particular to the WEPC.

5.3 Wolfram Enterprise Private Cloud

The Wolfram Enterprise Private Cloud (WEPC) is a self-contained computational ecosystem that handles all types of Wolfram Language evaluations. One of its main purposes for SYLVA is the deployment and rendering of CloudObjects. From previous sections we learned how CloudObjects are used in courseware to embed interactive elements into lectures. In assessments they are used for the question interfaces. While not all questions contain interactive elements, Wolfram Language specific elements such as Interpreters require access to the WEPC.

To deploy, host, and serve CloudObjects the WEPC requires a complex software architecture including its own user management, file storage, database, application server (Tomcat[3]), and kernels, to perform computations. In contrast to the components we discussed in the last section, the WEPC is not a microservice. It is delivered as a virtual machine that can be launched on any server running CentOS. The configuration of the WEPC involves—beside machine and network configurations—more than 200 specific parameters. Most important for SYLVA is the kernel allocation. There are four kernel types: session kernels, deployment kernels, task kernels, and data paclet kernels (Wolfram Research, 2019). Since the cloud is not accessed directly by end users through the cloud interface—it is rather a compute engine that runs in the background—mostly, deployment kernels are needed. Task kernels enable the automation of (recurring) scheduled or continuous computations, which is not relevant for SYLVA. Data paclet kernels, however, are needed to provide knowledge resources such

[3]Apache Tomcat® is an open source software used to power web applications, using Java technologies (The Apache Software Foundation, 2019).

as Entity lookups or CityData—as used in Example Question 3 in Section 3.4.2, for instance. Since kernels are competing for computational resources the allocation has a major impact on the performance with regard to the services that are using those kernels. Other than licensing restrictions, there are no upper boundaries on the total number of kernels that can be allocated. When, however, several kernels share one CPU the performance per kernel decreases. An ILAP like SYLVA demands—depending on the number of students and the type of questions—a large number of kernels in peak load situations such as in-class exams. When a student interacts with a question, for example, a kernel is launched and reserved for as long as it is actively used for computations. Typically, student learning and assessment activity will scatter over time. Critical, however, are situations where student activity is concentrated (in terms of time) and intense (computationally). This happens when specimens are created for an exam. Once the number of requests exceeds the number of kernels, additional requests will queue up. If the queuing limit is reached, the system becomes fragile and crashes can occur. This worst case scenario can cause significant harm—even the cancellation of the exam—and is, therefore, to be prevented by all means. This requires significant engineering effort for two reasons: First, exams are difficult to simulate since the computational resources required depend on the content and cannot be estimated easily ex ante. Simulations are technically challenging as all questions are embedded as iframes and assessments require independent users and authentication. Second, the computational infrastructure is expensive since the WEPC cannot be scaled down like a microservice when the load goes down. The second problem will decrease in severity when the utilization becomes less volatile—that is, when more of the exams are taken on different dates and in different time zones. With growing numbers of users, however, several connected WEPC instances must be run, which requires additional engineering.

From the previous discussion it becomes clear that the technical effort required for this setup is only worthwile for larger numbers of educators using the platform—for individual educators it is not. Note that for scientific computations that are required for an ILAP a specific computational engine is needed because common web technologies do not support such computations. While there are alternative open-source technologies such as jupyter, they lack the deep level of integration and consistency of the Wolfram Language (Somers, 2018). The WEPC supports various other programming languages, such as Python, R, and Node.js through ExternalEvaluate and libraries such as RLink.

Although the WEPC is not a microservice it can be used to create those type of applications through APIFunctions that are hosted on the WEPC. SYLVA uses such functions for the grading, deployment, and the preview of questions in CREO. This enables enhancements in several directions: First, the APIFunctions can be used for (user-) customized AI graders in fields such as image, voice, and natural language processing. These grading APIs can be easily used as in-

quiry conditions. Second, with APIFunctions the services provided by SYLVA can be integrated into other educational platforms, including LMSs, by adopting technical standards such as LTI (IMS Global, 2019).

CREO itself does not run on the WEPC but uses it is as its CloudBase for deployments and all processes afterward. This allows users to work with CREO locally and offline. All the computations that are performed during the authoring process, therefore, do not consume computational resources from the WEPC. Since CREO is fully based on the Wolfram Language it would be possible to turn it into a native web application, but currently the Wolfram Cloud does not support all the features of the Wolfram Language.

5.4 Roles and authentication

To account for different use cases and collaboration arrangements, SYLVA offers granular permissions using a role-based access control (RBAC) approach. In this way, each user gets assigned a role—either by invitation or via signup. To preclude misuse users need to be authorized for their roles by other users (invitations), or by a platform administrator (signup). Each role includes a predefined set of permissions that allow users to execute certain actions in SYLVA. These sets, for each role, consist of hundreds of low-level permissions, including—for example—fetch a list of all lectures, create a lecture, or delete a lecture. In Section 4.1 we explained the administration tool via which all these permissions are documented and can be simulated.

Every user can have multiple roles. This enables scenarios where a given user is a student in one course, and a teaching assistant in another course, for instance. For such cases, we need to distinguish two dimensions of roles: *system roles* and *project roles*.

System roles

System roles give access to specific actions on the web platform (SYLVA) and with regard to authoring features (CREO). In this dimension, four roles are distinguished:

- sylvauser
  The most basic role that is required to access the SYLVA platform. It is assigned by default to any invited user—students and educators.
- creouser
  This role is required for authoring with CREO, to create, view, and edit courses and files, and to send proposals. By default it is assigned to educators.
- developer
  This role contains the permissions of sylvauser and creouser, plus specific

features for maintenance and development purposes, such as advanced deployment and testing options and additional file operations.

- superadmin
  Combines the permissions of all other roles, and access to all projects and files. Only this role allows one to create organizations and users.

The *superadmin* and *developer* roles are only assigned to platform administrators and SYLVA developers, to protect the privacy of user data. The *creouser* role is not assigned to students since they are not involved in the authoring process—unless they are teaching assistants in other courses. Note that the *sylvauser* role only grants access to the platform, but not to particular courses (projects). A student—to be enrolled in a course—therefore also needs a *project role*.

Project roles

Project roles are assigned at the course level and define the hierarchy of permissions as follows:

- owner
  Has all permissions necessary to create, edit, view, and delete course contents, including the course itself. In addition, the owner can invite users to the project. This role is, by default, assigned to the user who creates a project.

- lecturer
  The lecturer role includes all the permissions of owners, except those necessary to delete the project. Lecturers are shown in the course syllabus.

- teaching assistant
  Teaching assistants can invite students, create course content, and send proposals. In contrast to lecturers they cannot accept proposals or publish grades.

- student
  Students can access all course materials and assessments, including corresponding evaluations. Students do not have access to the Administration app.

With a combination of system roles and project roles the most common use cases of collaboration, teaching, and learning in higher education can be realized. The RBAC approach makes it possible to add more roles over time to account for more granular permissions.

Security and authorization

SYLVA uses modern security standards to protect user data, all communication between users and the platform, and the platform itself. The core resembles a 3-tier architectural pattern:

1. Front end layer (web applications)
2. Back end layer (server API and microservices)
3. Database layer

The communication between the tiers as well as every user request (action) is protected by means of encryption and tokenization. In addition, the database layer is hosted in an internal network that is only accessible by the back end layer. As a consequence it is protected from external threats. The first two layers are auto-scalable and served through secure (HTTPS and TLS) proxy servers to counteract any potential DoS (Denial of Service) attack.

Bibliography

Adzharuddin, N. A. and L. H. Ling (2013): "Learning Management System (LMS) Among University Students: Does It Work?" *International Journal of e-Education, e-Business, e-Management and e-Learning*, 3, 248–252.

AESG (2019): "SYLVA Explained," https://www.sylva.ac/product/tutorials.

Amazon (2019): "Amazon Alexa," https://developer.amazon.com/alexa.

Apple Inc. (2019): "Siri - Apple," https://www.apple.com/siri/.

Arnold, I. J. (2016): "Cheating at Online Formative Tests: Does it Pay Off?" *Internet and Higher Education*, 29, 98–106.

Arvan, L. (2009): "Dis-Integrating the LMS," *Educause Quarterly*, 32.

Aydin, C. C. and G. Tirkes (2010): "Open Source Learning Management Systems in e-Learning and Moodle," *Proceedings of 2010 IEEE Education Engineering Conference, EDUCON 2010*, 593–600.

Babcock, P. S. (2010): "Real costs of nominal grade inflation? New evidence from student course evaluations." *Economic Inquiry*, 48, 983–996.

Baker, R., B. Evans, and T. Dee (2016): "A Randomized Experiment Testing the Efficacy of a Scheduling Nudge in a Massive Open Online Course (MOOC)," *AERA Open*, 2, 1–18.

Baleni, Z. G. (2015): "Online Formative Assessment in Higher Education: Its Pros and Cons," *The Electronic Journal of e-Learning*, 13, 228–236.

Becker, W. E. and C. Johnston (1999): "The Relationship Between Multiple Choice and Essay Response Questions in Assessing Economics Understanding," *The Economic Record*, 75, 348–357.

Bettinger, E. P., L. Fox, S. Loeb, and E. S. Taylor (2017): "Virtual Classrooms: How Online College Courses Affect Student Success," *American Economic Review*, 107, 2855–2875.

Brookhart, S. M., T. R. Guskey, A. J. Bowers, J. H. McMillan, and J. K. Smith (2016): "A Century of Grading Research: Meaning and Value in the Most Common Educational Measure," *Review of Educational Research*, 86, 803–848.

Bundy, A. (2007): "Computational Thinking is Pervasive," *Journal of Scientific and Practical Computing*, 1, 67–69.

Calderon, A. (2018): "Massification of higher education revisited," Tech. rep., Melbourne.

Cassels, J. R. T. and A. H. Johnstone (1984): "The Effect of Language on Student Performance on Multiple Choice Tests in Chemistry," *Journal of Chemical Education*, 61, 613–615.

Chen, J. C., D. C. Whittinghill, and J. A. Kadlowec (2010): "Classes that Click: Fast Rich Feedback to Enhance Student Learning and Satisfaction," *Journal of Engineering Education*, 99, 159–168.

Chung, C.-H., L. A. Pasquini, and C. E. Koh (2013): "Web-Based Learning Management System Considerations for Higher Education," *Learning and Performance Quarterly*, 1, 24–37.

Clio, M. (2003): "Grading on my Nerves," *The Chronicle of Higher Education*.

Coates, H., R. James, and G. Baldwin (2005): "A Critical Examination of the Effects of Learning Management Systems on University Teaching and Learning," *Tertiary Education and Management*, 11, 19–36.

Dahlstrom, E., D. C. Brooks, and J. Bichsel (2014): "The Current Ecosystem of Learning Management Systems in Higher Education: Student, Faculty, and IT Perspectives," Tech. rep., Educause Center for Analysis and Research.

Dalsgaard, C. (2006): "Social Software: E-Learning Beyond Learning Management Systems," *European Journal of Open, Distance and E-Learning*, 2.

DigitalEd (2019): "Möbius Assessment Question Types," https://www.digitaled.com/products/assessment/ques.

Docebo (2016): "eLearning Market Trends and Forecast 2017-2021," Tech. rep.

Euler, L. (1822): *Elements of Algebra*, London: Longman, Hurst, Rees, Orme, and Co., 3 ed.

Favreau, M. and N. S. Segalowitz (1982): "Second Language Reading in Fluent Bilinguals," *Applied Psycholinguistics*, 3, 329–341.

Fox, A., D. Patterson, S. Joseph, and P. Mcculloch (2015): "MAGIC: Massive Automated Grading in the Cloud," *Proceedings of Trends in Digital Education: Selected Papers from EC-TEL 2015 Workshops CHANGEE, WAPLA, and HybridEd*, 39–50.

Gaytan, J. and B. C. McEwen (2007): "Effective Online Instructional and Assessment Strategies." *American Journal of Distance Education*, 21, 117–132.

Geigle, C., C. Zhai, and D. C. Ferguson (2016): "An Exploration of Automated Grading of Complex Assignments," *Proceedings of the 3rd ACM Conference on Learning at Scale*, 351–360.

Gikandi, J. W., D. Morrow, and N. E. Davis (2011): "Online Formative Assessment in Higher Education: A Review of the Literature," *Computers & Education*, 57, 2333–2351.

Hastings, C., K. Mischo, and M. Morrison (2015): *Hands-On Start to Wolfram Mathematica and Programming with the Wolfram Language*, Champaign: Wolfram Media.

Huang, C. J. and P. S. Crooke (1997): *Mathematics and Mathematica for Economists*, Oxford: Blackwell Publishers, 1 ed.

Huba, M. E. and J. E. Freed (2000): *Learner-Centered Assessment on College Campuses: Shifting the Focus from Teaching to Learning*, Needham Heights: Addyn and Bacon.

IMS Global (2019): "Learning Tools Interoperability Core Specification," *IMS Final Release Version 1.3*, https://www.imsglobal.org/spec/lti/v1p3/.

Kirschner, F., F. Paas, and P. A. Kirschner (2009): "A Cognitive Load Approach to Collaborative Learning: United Brains for Complex Tasks," *Educational Psychology Review*, 21, 31–42.

Kroner, G. (2014): "Does your LMS do this?" *Edutechnica*, https://edutechnica.com/2014/01/07/a–model–for–lms.

Krumboltz, J. D. and C. J. Yeh (1996): "Competitive Grading Sabotages Good Teaching," *The Phi Delta Kappan*, 78, 324–326.

Kunnan, A. J. (2013): "Fairness and Justice in Language Assessment: Principles and Public Reasoning," *2013 CELC Conference at NUS, Singapore*, 36–39.

MathJax (2019): "MathJax," https://www.mathjax.org/.

MongoDB (2019a): "MongoDB Atlas," https://www.mongodb.com/cloud/atlas.

——— (2019b): "The Database for Modern Applications," https://www.mongodb.com/.

Nagel, R. (1995): "Unraveling in Guessing Games: An Experimental Study," *The American Economic Review*, 85, 1313–1326.

Ouadoud, M., A. Nejjari, M. Y. Chkouri, and K. E. El-Kadiri (2018): "Learning Management System and the Underlying Learning Theories: Towards a New Modeling of an LMS," *International Journal of Information Science & Technology*, 2, 25–33.

Rapaport, W. J. (2011): "A Triage Theory of Grading: The Good, the Bad, and the Middling," *Teaching Philosophy*, 34, 347–372.

Rojstaczer, S. and C. Healy (2012): "Where A is Ordinary: The Evolution of American College and University Grading, 1940-2009," *Teachers College Record*, 114, 1–23.

Scager, K., J. Boonstra, T. Peeters, J. Vulperhorst, and F. Wiegant (2016): "Collaborative Learning in Higher Education: Evoking Positive Interdependence," *CBE - Life Sciences Education*, 15, 1–9.

Shermis, M. D. (2014): "State-of-the-Art Automated Essay Scoring: Competition, Results, and Future Directions from a United States Demonstration," *Assessing Writing*, 20, 53–76.

Slavin, R. E. (1980): "Cooperative Learning," *Review of Educational Research*, 50, 315–342.

——— (2014): "Cooperative Learning and Academic Achievement: Why Does Groupwork Work?" *Anales de Psicología*, 30, 785–791.

Smith, A. (2012): *The Wealth of Nations*, Herts: Wordsworth Editions Ltd.

Smith, G. (2007): "How Does Student Performance on Formative Assessments Relate to Learning Assessed by Exams?" *Journal of College Science Teaching*, 36, 28–34.

Somers, J. (2018): "The Scientific Paper Is Obsolete Here's What's Next." *The Atlantic*.

The Apache Software Foundation (2019): "Apache Tomcat®," http://tomcat.apache.org/.

Trott, M. (2004): *The Mathematica GuideBook for Programming*, New York: Springer Verlag.

Tversky, A. and D. Kahneman (1981): "The Framing of Decisions and the Psychology of Choice," *Science*, 211, 453–458.

Van Dalen, R. C., K. M. Knill, and M. J. F. Gales (2015): "Automatically Grading Learners' English Using a Gaussian Process," *Proceedings of SLaTe 2015*, 7–12.

Varian, H. R. (1992): *Economic and Financial Modeling with Mathematica®*, New York: Springer-Verlag/TELOS.

——— (2009): *Intermediate Microeconomics: A Modern Approach*, New York: W. W. Norton & Company, 8 ed.

von Humboldt, W. (1997): *Bildung und Sprache*, Paderborn: Ferdinand Schöningh, 5 ed.

Vrasidas, C. (2004): "Issues of Pedagogy and Design in e-Learning Systems," *Proceedings of the 2004 ACM Symposium on Applied Computing - SAC 2004*, 911–915.

Wilcox, C. (2016): "Testing Strategies for the Automated Grading of Student Programs," *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE 2016*, 437–442.

Wing, J. M. (2006): "Computational Thinking," *Communications of the ACM*, 49, 33–35.

——— (2008): "Computational Thinking and Thinking about Computing," *Philosophical Transactions of the Royal Society A*, 366, 3717–3725.

Wolff, R. P. (1969): "A Discourse on Grading, Part 2," in *The Ideal of the University*, Boston: Beacon Press, chap. 1, 58–68.

Wolfram, S. (2016a): *An Elementary Introduction to the Wolfram Language*, Champaign: Wolfram Media.

——— (2016b): "How to Teach Computational Thinking," *Stephen Wolfram Blog*, https://blog.stephenwolfram.com/2016/09/how–to–tea.

Wolfram Research (2019): "Wolfram Enterprise Private Cloud Documentation," https://www.wolframcloud.com/objects/documentation.

Yueh, H.-P. and H. Shihkuan (2008): "Designing a Learning Management System to Support Instruction," *Communications of the ACM*, 51, 59–63.

Zhang, M. (2013): "Contrasting Automated and Human Scoring of Essays," *R\&D Connections*, 21, 1–11.

Curriculum Vitae of Maik Meusel

Personal information

Citizenship	Germany
Birth date	25.12.1984 in Berlin, Germany

Education

6/2013 — 10/2019	PhD Candidate in Business Administration University of Zurich, Zurich, Switzerland Advisor Prof. Karl Schmedders Chair for Quantitative Business Administration
07/2019 — 08/2019	New Initiative for Computational Economics Stanford University, Stanford (CA), USA
11/2017 — 01/2018	Visiting Fellow, The Hoover Institution Stanford University, Stanford (CA), USA
06/2016 — 07/2016	Wolfram Summer School Bentley University, Waltham (MA), USA
02/2009 — 03/2013	M.A. in Economics University of Zurich, Zurich, Switzerland
10/2005 — 10/2008	B.Sc. in Economics Humboldt University of Berlin, Berlin, Germany